LPI exam 101 prep: Devices, Linux filesystems, and the Filesystem Hierarchy Standard

Junior Level Administration (LPIC-1) topic 104

Skill Level: Intermediate

Ian Shields (ishields@us.ibm.com)
Senior Programmer
IBM

28 Dec 2005

Updated 16 Apr 2006

In this tutorial, Ian Shields continues preparing you to take the Linux Professional Institute® Junior Level Administration (LPIC-1) Exam 101. In this fourth in a series of five tutorials, Ian introduces you to Linux® devices, filesystems, and the Filesystem Hierarchy Standard. By the end of this tutorial, you will know how to create and format partitions with different Linux filesystems and how to manage and maintain those systems.

Section 1. Before you start

Learn what these tutorials can teach you and how you can get the most from them.

About this series

The Linux Professional Institute (LPI) certifies Linux system administrators at two levels: *junior level* (also called "certification level 1") and *intermediate level* (also called "certification level 2"). To attain certification level 1, you must pass exams 101 and 102; to attain certification level 2, you must pass exams 201 and 202.

developerWorks offers tutorials to help you prepare for each of the four exams. Each exam covers several topics, and each topic has a corresponding self-study tutorial on developerWorks. For LPI exam 101, the five topics and corresponding

developerWorks tutorials are:

Table 1. LPI exam 101: Tutorials and topics

LPI exam 101 topic	developerWorks tutorial	Tutorial summary
Topic 101	LPI exam 101 prep (topic 101): Hardware and architecture	Learn to configure your system hardware with Linux. By the end of this tutorial, you will know how Linux configures the hardware found on a modern PC and where to look if you have problems.
Topic 102	LPI exam 101 prep: Linux installation and package management	Get an introduction to Linux installation and package management. By the end of this tutorial, you will know how Linux uses disk partitions, how Linux boots, and how to install and manage software packages.
Topic 103	LPI exam 101 prep: GNU and UNIX commands	Get an introduction to common GNU and UNIX commands. By the end of this tutorial, you will know how to use commands in the bash shell, including how to use text processing commands and filters, how to search files and directories, and how to manage processes.
Topic 104	LPI exam 104 prep: Devices, Linux filesystems, and the Filesystem Hierarchy Standard.	(This tutorial). Learn how to create filesystems on disk partitions, as well as how to make them accessible to users, manage file ownership and user quotas, and repair filesystems as needed. Also learn about hard and symbolic links, and how to locate files in your filesystem and where files should be placed. See detailed objectives below.
Topic 110	LPI exam 110 prep: The X Window system	Learn about the X Window System on Linux. By the end of this tutorial, you will know how to install and maintain the X Window System. This tutorial covers both major packages for X on Linux: XFree86 and X.Org.

To pass exams 101 and 102 (and attain certification level 1), you should be able to:

- Work at the Linux command line

- Perform easy maintenance tasks: help out users, add users to a larger system, back up and restore, and shut down and reboot
- Install and configure a workstation (including X) and connect it to a LAN, or connect a stand-alone PC via modem to the Internet

To continue preparing for certification level 1, see the developerWorks tutorials for LPI exam 101. Read more about the entire set of developerWorks LPI tutorials.

The Linux Professional Institute does not endorse any third-party exam preparation material or techniques in particular. For details, please contact info@lpi.org.

About this tutorial

Welcome to "Devices, Linux filesystems, and the Filesystem Hierarchy Standard", the fourth of five tutorials designed to prepare you for LPI exam 101. In this tutorial, you learn how to create and manage partitions. You also learn about the *Filesystem Hierarchy Standard (FHS)*, which recommends where different types of data can be found and should be stored on a Linux system.

This tutorial is organized according to the LPI objectives for this topic. Very roughly, expect more questions on the exam for objectives with higher weight.

Table 2. Devices, Linux filesystems, and the Filesystem Hierarchy Standard: Exam objectives covered in this tutorial

LPI exam objective	Objective weight	Objective summary
1.104.1 Create partitions and filesystems	Weight 3	Learn to configure disk partitions and create filesystems on media such as hard disks. Use various `mkfs` commands to set up filesystems, including ext2, ext3, reiserfs, vfat, and xfs.
1.104.2 Maintain the integrity of filesystems	Weight 3	Learn to verify the integrity of filesystems, monitor free space and inodes, and repair simple filesystem problems. Learn to maintain standard filesystems and journaling filesystems.
1.104.3 Mount and unmount filesystems	Weight 3	Learn to mount and unmount filesystems manually. Also learn to configure filesystem mounting on bootup, and configure removable filesystems, such as tape drives, floppies, and CDs, so that ordinary users can mount and unmount them.
1.104.4	Weight 5	Learn to manage disk quotas

Manage disk quota		for users, including setting up quotas for a filesystem, and editing, checking, and generating user quota reports.
1.104.5 Use file permissions to control access to files	Weight 5	Learn to control file access through permissions, including access permissions on regular and special files as well as directories. Also learn about access modes such as suid, sgid, and the sticky bit; the use of the group field to grant file access to workgroups; the immutable flag; and the default file creation mode.
1.104.6 Manage file ownership	Weight 1	Learn to control user and group ownership of files, including how to change the user and group owner of a file as well as the default group owner for new files.
1.104.7 Create and change hard and symbolic links	Weight 1	Learn to create and manage hard and symbolic links to a file, including creating and identifying links. Learn to copy files through links, and use linked files to support system administration tasks.
1.104.8 Find system files and place files in the correct location	Weight 5	Learn about the Filesystem Hierarchy Standard, including typical file locations and directory classifications. Learn to find files and commands on a Linux system.

Prerequisites

To get the most from this tutorial, you should have a basic knowledge of Linux and a working Linux system on which you can practice the commands covered in this tutorial.

This tutorial builds on content covered in the previous three tutorials in this series, so you may want to first review the tutorials for topics 101, 102, and 103.

Different versions of a program may format output differently, so your results may not look exactly like the listings and figures in this tutorial.

Section 2. Creating partitions and filesystems

This section covers material for topic 1.104.1 for the Junior Level Administration (LPIC-1) exam 101. The topic has a weight of 3.

In this section, you learn how to:

- Configure disk partitions
- Create filesystems on hard disks and other media
- Use `mkfs` commands to set up ext2, ext3, reiserfs, vfat, and xfs partitions

First, a quick review. In the tutorial for topic 101, "LPI exam 101 prep (topic 101): Hardware and architecture," you learned about IDE and SCSI hard drives such as /dev/hda and /dev/sdb, and partitions on these drives, such as /dev/hda1, /dev/hda5 and /dev/sda1.

In the tutorial for topic 102, "LPI exam 101 prep (topic 102): Linux installation and package management," you learned more about partitions, including *primary*, *extended*, and *logical* partitions. You also learned that a Linux filesystem contains *files* that are arranged on a disk or other *block storage device* in *directories*. As with many other systems, directories on a Linux system may contain other directories called *subdirectories*. That tutorial also discussed the considerations that guide you in making choices about partitioning.

This section reviews block devices and partitions, and then introduces you to the `fdisk` command, which is used to create, modify, or delete partitions on block devices. It also introduces the various forms of the `mkfs` command (mkfs stands for *make filesystem*); these commands are used to format partitions as a particular filesystem type.

Note: In addition to the tools and filesystems required for the LPI exams, you may encounter or need other tools and filesystems. Find a brief summary of some other available tools in Other tools and filesystems.

Block devices and partitions

Let's quickly review block devices and partitions. If you need more information, refer back to the tutorials for topic 101 and topic 102.

Block devices

A *block device* is an abstraction layer for any storage device that can be formatted in fixed-size *blocks*; individual blocks may be accessed independently of access to other blocks. Such access is often called *random access*.

The abstraction layer of randomly accessible fixed-size blocks allows programs to use these block devices without worrying about whether the underlying device is a hard drive, floppy, CD, network drive, or some type of virtual device such as an in-memory file system.

Examples of block devices include the first IDE hard drive on your system (/dev/hda) or the second SCSI drive (/dev/sdb). Use the `ls -l` command to display /dev entries. The first character on each output line is **b** for a **block** device, such as floppy, CD drive, IDE hard drive, or SCSI hard drive; and **c** for a **character** device, such as a tape drive or terminal. See the examples in Listing 1.

Listing 1. Linux block and character devices

```
[ian@lyrebird ian]$ ls -l /dev/fd0 /dev/hda /dev/sdb /dev/st0 /dev/tty0
brw-rw----    1 ian      floppy     2,   0 Jun 24  2004 /dev/fd0
brw-rw----    1 root     disk       3,   0 Jun 24  2004 /dev/hda
brw-rw----    1 root     disk       8,  16 Jun 24  2004 /dev/sdb
crw-rw----    1 root     disk       9,   0 Jun 24  2004 /dev/st0
crw--w----    1 root     root       4,   0 Jun 24  2004 /dev/tty0
```

Partitions

For some block devices, such as floppy disks and CD or DVD discs, it is common to use the whole media as a single filesystem. However, with large hard drives, and even with smaller USB memory keys, it is more common to divide, or partition, the available space into several different *partitions*.

Partitions can be different sizes, and different partitions may have different filesystems on them, so a single disk can be used for many purposes, including sharing it between multiple operating systems. For example, I use test systems with several different Linux distributions and sometimes a Windows® system, all sharing one or two hard drives.

You will recall from the 101 and 102 tutorials that hard drives have a *geometry*, defined in terms of cylinders, heads, and sectors. Even though modern drives use *logical block addressing* (*LBA*), which renders geometry largely irrelevant, the fundamental allocation unit for partitioning purposes is still the cylinder.

Displaying partition information

Partition information is stored in a *partition table* on the disk. The table lists information about the start and end of each partition, information about its *type*, and whether it is marked bootable or not. To create and delete partitions, you edit the partition table using a program specially designed for the job. For the LPI exam, you need to know about the `fdisk` program, so that is what is covered here, although several other tools exist.

The `fdisk` command with the `-l` option is used to list partitions. Add a device name, such as /dev/hda, if you want to look at the partitions on a particular drive. Note that partitioning tools require root access. Listing 2 shows the partitions on one of my hard drives.

Listing 2. Listing partitions with fdisk

```
[root@lyrebird root]# fdisk -l /dev/hda

Disk /dev/hda: 160.0 GB, 160041885696 bytes
255 heads, 63 sectors/track, 19457 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot    Start       End    Blocks   Id  System
/dev/hda1   *         1      2078  16691503+   7  HPFS/NTFS
/dev/hda2          2079      3295   9775552+   c  Win95 FAT32 (LBA)
/dev/hda3          3296      3422   1020127+  83  Linux
/dev/hda4          3423     19457 128801137+   f  Win95 Ext'd (LBA)
/dev/hda5          3423      3684   2104483+  82  Linux swap
/dev/hda6          3685      6234  20482843+  83  Linux
/dev/hda7          6235      7605  11012526   83  Linux
/dev/hda8          7606      9645  16386268+  83  Linux
/dev/hda9          9646     12111  19808113+  83  Linux
/dev/hda10        12112     15680  28667961   83  Linux
/dev/hda11        15681     19457  30338721   83  Linux
```

Notes:

1. The header information shows the disk size and geometry. Most large disks using LBA have 255 heads per cylinder and 63 sectors per track, making a total of 16065 sectors, or 8225280 bytes per cylinder.

2. In this example, the first primary partition (/dev/hda1) is marked *bootable* (or *active*). As you saw in the tutorial for topic 102, this enables the standard DOS PC master boot record to boot the partition. This flag has no significance for the LILO or GRUB boot loaders.

3. The *Start* and *End* columns show the starting and ending cylinder for each partition. These must not overlap and should generally be contiguous, with no intervening space.

4. The *Blocks* column shows the number of 1K (1024 byte) blocks in the partition. The maximum number of blocks in a partition is therefore half of the product of the number of cylinders (End + 1 - Start) and the number of sectors per cylinder. A trailing + sign indicates that not all sectors in the partition are used.

5. The *Id* field indicates the intended use of the partition. Type 82 is a Linux swap partition, and type 83 is a Linux data partition. There are approximately 100 different partition types defined. This particular disk is shared between several operating systems, including Windows/XP, hence the presence of Windows NTFS (and FAT32) partitions.

Partitioning with fdisk

You have just seen how to display partition information using the `fdisk` command. This command also provides an interactive environment for editing the partition table to create or remove partitions.

Warnings

Before you start modifying partitions, there are some important things to remember. You risk **losing your existing data** if you do not follow these guidelines.

1. **Do not change partitions that are in use**. Plan your actions and execute them carefully.

2. **Know your tool**. The `fdisk` command does not commit any changes to your disk until you tell it to. Other tools, including `parted`, may commit changes as you go.

3. **Back up important data before you start**, as with any operation that may cause data loss.

4. Partitioning tools write the partition table. Unless the tool you are using also includes the ability to move, resize, format, or otherwise write to the data area of your disk, your data will not be touched. If you do make an accidental mistake, stop as quickly as possible and seek help. You may still be able to recover your partitions and data.

Start fdisk

To start `fdisk` in interactive mode, simply give the name of a disk, such as /dev/hda or /dev/sdb, as a parameter. The following example boots a Knoppix live CD. You will need root authority, and you will see output similar to Listing 3.

Listing 3. Starting interactive fdisk

```
root@ttyp1[knoppix]# fdisk  /dev/hda

The number of cylinders for this disk is set to 14593.
There is nothing wrong with that, but this is larger than 1024,
and could in certain setups cause problems with:
1) software that runs at boot time (e.g., old versions of LILO)
2) booting and partitioning software from other OSs
   (e.g., DOS FDISK, OS/2 FDISK)

Command (m for help):
```

Most modern disks have more than 1024 cylinders, so you will usually see the warning as shown in Listing 3. Type `m` to display a list of available one-letter commands as shown in Listing 4.

Listing 4. Help in fdisk

```
Command action
   a   toggle a bootable flag
   b   edit bsd disklabel
   c   toggle the dos compatibility flag
   d   delete a partition
   l   list known partition types
   m   print this menu
   n   add a new partition
   o   create a new empty DOS partition table
   p   print the partition table
```

```
   q   quit without saving changes
   s   create a new empty Sun disklabel
   t   change a partition's system id
   u   change display/entry units
   v   verify the partition table
   w   write table to disk and exit
   x   extra functionality (experts only)

Command (m for help):
```

Use the p command to display the existing partition on this particular disk; Listing 5 shows the output.

Listing 5. Displaying the existing partition table

```
Disk /dev/hda: 120.0 GB, 120034123776 bytes
255 heads, 63 sectors/track, 14593 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hda1   *           1        2611    20972826    7  HPFS/NTFS

Command (m for help):
```

This particular disk is a 120GB disk with a Windows/XP partition of approximately 20GB. It is a primary partition, and it is marked bootable, as is typical for a Windows system.

Our workstation layout

Let's now use part of the free space to set up a simple workstation with the following additional partitions. In practice, it's unlikely that you would mix this many different filesystem types, but we'll do it here for illustration purposes.

1. Another primary partition for our boot files. This will be mounted as /boot and will contain kernel files and initial ramdisk files. If you use the GRUB boot loader, GRUB files will also be located here. From the tutorial for topic 102, our guideline is for approximately 100MB. We see from Listing 5 that a cylinder contains approximately 8MB of data, so we will use 13 cylinders for /boot. This will be /dev/hda2.
2. We will create an extended partition to house logical partitions spanning the rest of the free space. This will be /dev/hda3.
3. We will create a 500MB swap partition as /dev/hda5. We will use 64 cylinders for this.
4. We will create a logical partition of approximately 20GB for our Linux system. This will be /dev/hda6.
5. We will create a separate 10GB partition for user data. This will eventually be mounted as /home. For now, it will simply be /dev/hda7.

6. Finally, we will create a small 2GB partition for sharing data between the Linux and Windows systems. This will eventually be formatted as FAT32 (or vfat). This will be /dev/hda8.

Creating our partitions

Let's start by using the n command to create a new partition; see Listing 6.

Listing 6. Creating our first partition

```
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 2
First cylinder (2612-14593, default 2612):
Using default value 2612
Last cylinder or +size or +sizeM or +sizeK (2612-14593, default 14593): 2624

Command (m for help): p

Disk /dev/hda: 120.0 GB, 120034123776 bytes
255 heads, 63 sectors/track, 14593 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hda1   *           1        2611    20972826    7  HPFS/NTFS
/dev/hda2            2612        2624      104422+  83  Linux

Command (m for help):
```

We took the default for the first cylinder and specified the value of 2624 for the last cylinder, resulting in a 13-cylinder partition. You can see from Listing 6 that our partition is, indeed, approximately 100MB in size. Since it is a primary partition, it must be numbered from 1 through 4. It is a good idea to assign partition numbers sequentially; some tools complain if this is not done.

Notice also that our new partition was assigned a type of 83, for a Linux data partition. Think of this as an indicator to the operating system of the intended use of the partition. The eventual use should match this, but at this point we don't even have the partition formatted, let alone have any data on it.

We now create the extended partition, which is a container for the logical partitions on the disk. We assign partition number 3 (/dev/hda3). The interaction and result is shown in Listing 7. Note again that the partition type was assigned automatically.

Listing 7. Creating an extended partition

```
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
e
Partition number (1-4): 3
First cylinder (2625-14593, default 2625):
```

```
Using default value 2625
Last cylinder or +size or +sizeM or +sizeK (2625-14593, default 14593):
Using default value 14593

Command (m for help): p

Disk /dev/hda: 120.0 GB, 120034123776 bytes
255 heads, 63 sectors/track, 14593 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hda1   *           1        2611    20972826    7  HPFS/NTFS
/dev/hda2            2612        2624      104422+  83  Linux
/dev/hda3            2625       14593    96140992+   5  Extended

Command (m for help):
```

Now let's move on to creating a swap partition as a logical partition within our extended partition. We will use a value of +64 (cylinders) for the last cylinder, rather than performing the arithmetic ourselves. Note that this time we use the `t` command to assign a type of 82 (Linux swap) to the newly created partition. Otherwise, it would be another type 83 (Linux data) partition.

Listing 8. Creating a swap partition

```
Command (m for help): n
Command action
   l   logical (5 or over)
   p   primary partition (1-4)
l
First cylinder (2625-14593, default 2625):
Using default value 2625
Last cylinder or +size or +sizeM or +sizeK (2625-14593, default 14593): +64

Command (m for help): t
Partition number (1-5): 5
Hex code (type L to list codes): 82
Changed system type of partition 5 to 82 (Linux swap)

Command (m for help): p

Disk /dev/hda: 120.0 GB, 120034123776 bytes
255 heads, 63 sectors/track, 14593 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hda1   *           1        2611    20972826    7  HPFS/NTFS
/dev/hda2            2612        2624      104422+  83  Linux
/dev/hda3            2625       14593    96140992+   5  Extended
/dev/hda5            2625        2689      522081   82  Linux swap

Command (m for help):
```

Now let's define the main Linux partition and the /home partition. This time we will simply specify sizes of +20480M and +10240M, indicating 20GB and 10GB, respectively. We let `fdisk` calculate the number of cylinders for us. The results are shown in Listing 9.

Listing 9. Creating our main Linux partition

```
Command (m for help): n
Command action
   l   logical (5 or over)
```

```
   p   primary partition (1-4)
l
First cylinder (2690-14593, default 2690):
Using default value 2690
Last cylinder or +size or +sizeM or +sizeK (2690-14593, default 14593): +20480M

Command (m for help): n
Command action
   l   logical (5 or over)
   p   primary partition (1-4)
l
First cylinder (5181-14593, default 5181):
Using default value 5181
Last cylinder or +size or +sizeM or +sizeK (5181-14593, default 14593): +10240M

Command (m for help): p

Disk /dev/hda: 120.0 GB, 120034123776 bytes
255 heads, 63 sectors/track, 14593 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hda1   *           1        2611    20972826    7  HPFS/NTFS
/dev/hda2            2612        2624      104422+  83  Linux
/dev/hda3            2625       14593    96140992+   5  Extended
/dev/hda5            2625        2689      522081   82  Linux swap
/dev/hda6            2690        5180    20008926   83  Linux
/dev/hda7            5181        6426    10008463+  83  Linux

Command (m for help):
```

Our final partition is the FAT32 partition. We use the commands we have used above to create /dev/hda8 using a size specification of +2048M, and then we change the partition type to **b** (for a W95 FAT32 partition). Next we save all our work.

Saving our partition table

So far, we have just been doing an in-memory edit of a partition table. We could use the q command to quit without saving changes. If something is not as we like, it, we can use the d command to delete one or more partitions so we can redefine them. If we are happy with our setup, we use the v command to verify our setup, and then the w command to write the new partition table and exit. See Listing 10. If you run `fdisk -l` again, you will see that Linux now knows about the new partitions. Unlike in some operating systems, it is not always necessary to reboot to see the changes. A reboot may be required if, for example, /dev/hda3 became /dev/hda2 because the original /dev/hda2 was deleted. If a reboot is needed, `fdisk` should tell you to do so.

Listing 10. Saving the partition table

```
Command (m for help): v
127186915 unallocated sectors

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.

WARNING: If you have created or modified any DOS 6.x
partitions, please see the fdisk manual page for additional
information.
Syncing disks.
root@ttyp0[knoppix]# fdisk -l /dev/hda
```

```
Disk /dev/hda: 120.0 GB, 120034123776 bytes
255 heads, 63 sectors/track, 14593 cylinders
Units = cylinders of 16065 * 512 = 8225280 bytes

   Device Boot      Start         End      Blocks   Id  System
/dev/hda1   *           1        2611    20972826    7  HPFS/NTFS
/dev/hda2            2612        2624      104422+  83  Linux
/dev/hda3            2625       14593    96140992+   5  Extended
/dev/hda5            2625        2689      522081   82  Linux swap
/dev/hda6            2690        5180    20008926   83  Linux
/dev/hda7            5181        6426    10008463+  83  Linux
/dev/hda8            6427        6676     2008093+   b  W95 FAT32
```

More on fdisk

You may notice that we did not change the bootable flag on any partition. As our disk stands now, it still has the Windows Master Boot Record (MBR) and will therefore boot the first primary partition that is marked bootable (the NTFS partition in our example).

Neither LILO nor GRUB uses the bootable flag. If either of these is installed in the MBR, then it can boot the Windows/XP partition. You could also install LILO or GRUB into your /boot partition (/dev/hda2) and mark that partition bootable and remove the bootable flag from /dev/hda1. Leaving the original MBR can be useful if the machine is later returned to being a Windows-only machine.

You have now seen one way to set up a Linux workstation. Other choices you might make are covered later in this tutorial, under Finding and placing system files.

Filesystem types

Linux supports several different filesystems. Each has strengths and weaknesses and its own set of performance characteristics. One important attribute of a filesystem is *journaling*, which allows for much faster recovery after a system crash. Generally, a journaling filesystem is preferred over a non-journaling one when you have a choice. Following is a brief summary of the types you need to know about for the LPI exam. See Resources for additional background information.

The ext2 filesystem

The ext2 filesystem (also known as the *second extended filesystem*) was developed to address shortcomings in the Minix filesystem used in early versions of Linux. It has been used extensively on Linux for many years. There is no journaling in ext2, and it has largely been replaced by ext3.

The ext3 filesystem

The ext3 filesystem adds journaling capability to a standard ext2 filesystem and is therefore an evolutionary growth of a very stable filesystem. It offers reasonable performance under most conditions and is still being improved. Because it adds journaling on top of the proven ext2 filesystem, it is possible to convert an existing ext2 filesystem to ext3 and even convert back again if required.

The ReiserFS filesystem

ReiserFS is a B-tree-based filesystem that has very good overall performance, particularly for large numbers of small files. ReiserFS also scales well and has journaling.

The XFS filesystem

XFS is a filesystem with journaling. It comes with robust features and is optimized for scalability. XFS aggressively caches in-transit data in RAM, so an uninterruptible power supply is recommended if you use XFS.

The swap filesystem

Swap space must be formatted for use as swap space, but it is not generally considered a filesystem, otherwise.

The vfat filesystem

This filesystem (also known as *FAT32*) is not journaled and lacks many features required for a full Linux filesystem implementation. It is useful for exchanging data between Windows and Linux systems as it can be read by both Windows and Linux. Do **not** use this filesystem for Linux, except for sharing data between Windows and Linux. If you unzip or untar a Linux archive on a vfat disk, you will lose permissions, such as execute permission, and you will lose any symbolic links that may have been stored in the archive.

Both ext3 and ReiserFS are mature and used as the default filesystem on a number of distributions. These are recommended for general use.

Creating filesystems

Linux uses the `mkfs` command to create filesystems and `mkswap`command to make swap space. The `mkfs` command is actually a front end to several filesystem-specific commands such as `mkfs.ext3` for ext3 and `mkfs.reiserfs` for ReiserFS.

What filesystem support is already installed on your system? Use the `ls /sbin/mk*` command to find out. An example is shown in Listing 11.

Listing 11. Filesystem creation commands

```
root@ttyp0[knoppix]# ls /sbin/mk*
/sbin/mkdosfs      /sbin/mkfs.ext2   /sbin/mkfs.msdos     /sbin/mkraid
/sbin/mke2fs       /sbin/mkfs.ext3   /sbin/mkfs.reiserfs  /sbin/mkreiserfs
/sbin/mkfs         /sbin/mkfs.jfs    /sbin/mkfs.vfat      /sbin/mkswap
/sbin/mkfs.cramfs  /sbin/mkfs.minix  /sbin/mkfs.xfs
```

You will notice various forms of some commands. For example, you will usually find that the files mke2fs, mkfs.ext2 and mkfs.ext3 are identical, as are mkreiserfs and

mkfs.reiserfs.

There are a few common options for all `mkfs` commands. Options that are specific to the type of filesystem being created are passed to the appropriate creation command, based on the type of filesystem specified in the `-type` option. Our examples use `mkfs -type`, but you may use the other forms directly with equal effect. For example, you may use `mkfs -type reiserfs`, `mkreiserfs`, or `mkfs.reiserfs`. For the manual pages for a specific filesystem, use the appropriate `mkfs` command as the name, for example, `man mkfs.reiserfs`. Many of the values displayed in the output examples below can be controlled by options to `mkfs`.

Creating an ext3 filesystem

Listing 12. Creating an ext3 filesystem

```
root@ttyp0[knoppix]# mkfs -t ext3 /dev/hda8
mke2fs 1.35 (28-Feb-2004)
Filesystem label=
OS type: Linux
Block size=4096 (log=2)
Fragment size=4096 (log=2)
251392 inodes, 502023 blocks
25101 blocks (5.00%) reserved for the super user
First data block=0
16 block groups
32768 blocks per group, 32768 fragments per group
15712 inodes per group
Superblock backups stored on blocks:
        32768, 98304, 163840, 229376, 294912

Writing inode tables: done
Creating journal (8192 blocks): done
Writing superblocks and filesystem accounting information: done

This filesystem will be automatically checked every 32 mounts or
180 days, whichever comes first.  Use tune2fs -c or -i to override.
```

A useful option for ext2 and ext3 filesystems is the `-L` option with a name, which assigns a label to the partition. This can be used instead of the device name when mounting filesystems; it provides some level of insulation against changes that may need to be reflected in various control files. To display or set a label for an existing ext2 or ext3 filesystem, use the `e2label` command. Labels are limited to a maximum size of 16 characters.

Note that a journal is created with ext3. If you wish to add a journal to an existing ext2 system, use the `tune2fs` command with the `-j` option.

Creating a ReiserFS filesystem

Listing 13. Creating a ReiserFS filesystem

```
.root@ttyp0[knoppix]# mkfs -t reiserfs /dev/hda6
mkfs.reiserfs 3.6.17 (2003 www.namesys.com)

A pair of credits:
Many persons came to www.namesys.com/support.html,  and got a question answered
```

```
for $25, or just gave us a small donation there.

Jeremy  Fitzhardinge  wrote  the  teahash.c  code  for  V3.  Colin  Plumb  also
contributed to that.

Guessing about desired format. Kernel 2.4.26 is running.
Format 3.6 with standard journal
Count of blocks on the device: 5002224
Number of blocks consumed by mkreiserfs formatting process: 8364
Blocksize: 4096
Hash function used to sort names: "r5"
Journal Size 8193 blocks (first block 18)
Journal Max transaction length 1024
inode generation number: 0
UUID: 72e317d6-8d3a-45e1-bcda-ad7eff2b3b40
ATTENTION: YOU SHOULD REBOOT AFTER FDISK!
        ALL DATA WILL BE LOST ON '/dev/hda6'!
Continue (y/n):y
Initializing journal - 0%....20%....40%....60%....80%....100%
Syncing..ok

Tell your friends to use a kernel based on 2.4.18 or later, and especially not a
kernel based on 2.4.9, when you use reiserFS. Have fun.

ReiserFS is successfully created on /dev/hda6.
```

You can label a ReiserFS system using the `-l` (or `--label`option with a name). You can use the `reiserfstune` command to add a label or display the label on an existing ReiserFS filesystem. Labels are limited to a maximum size of 16 characters.

Creating an XFS filesystem

Listing 14. Creating an XFS filesystem

```
root@ttyp0[knoppix]# mkfs -t xfs /dev/hda7
meta-data=/dev/hda7              isize=256    agcount=16, agsize=156382 blks
         =                       sectsz=512
data     =                       bsize=4096   blocks=2502112, imaxpct=25
         =                       sunit=0      swidth=0 blks, unwritten=1
naming   =version 2              bsize=4096
log      =internal log           bsize=4096   blocks=2560, version=1
         =                       sectsz=512   sunit=0 blks
realtime =none                   extsz=65536  blocks=0, rtextents=0
```

You can label an XFS system using the `-L` option with a name. You can use the `xfs_admin` command with the `-L` option to add a label to an existing XFS filesystem. Use the `-l` option of `xfs_admin` to display a label. Unlike ext2, ext3 and ReiserFS, labels are limited to a maximum size of 12 characters.

Creating a vfat filesystem

Listing 15. Creating a vfat filesystem

```
root@ttyp0[knoppix]# mkfs -t vfat /dev/hda8
mkfs.vfat 2.10 (22 Sep 2003)
```

Label a FAT32 filesystems using the `-n` (for volume name) option. The `e2label` command will display or set the label on vfat as well as ext partitions. Labels on FAT32 are limited to a maximum size of 16 characters.

Creating swap space

Listing 16. Creating swap space

```
root@ttyp0[knoppix]# mkswap /dev/hda5
Setting up swapspace version 1, size = 534605 kB
```

Unlike regular filesystems, swap partitions aren't mounted. Instead, they are enabled using the `swapon` command. Your Linux system's startup scripts will take care of automatically enabling your swap partitions.

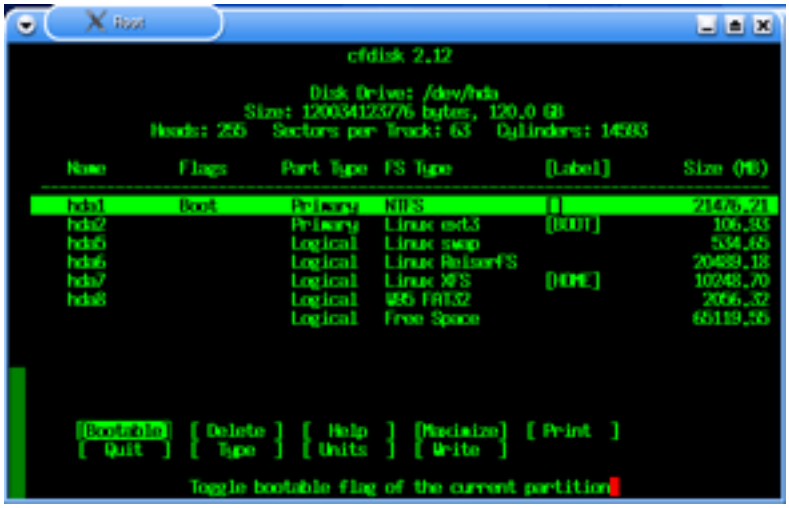
Other tools and filesystems

The following tools and filesystems are not part of the LPI objectives for this exam. This very brief overview touches on some of the tools and filesystems that you may encounter.

Partitioning tools

Many Linux distributions include the `cfdisk` and `sfdisk` commands. The `cfdisk` command provides a more graphical interface than fdisk, using the ncurses library functions as shown in Figure 1. The `sfdisk` command is intended for programmer use and can be scripted. Use it only if you know what you are doing.

Figure 1. Using cfdisk

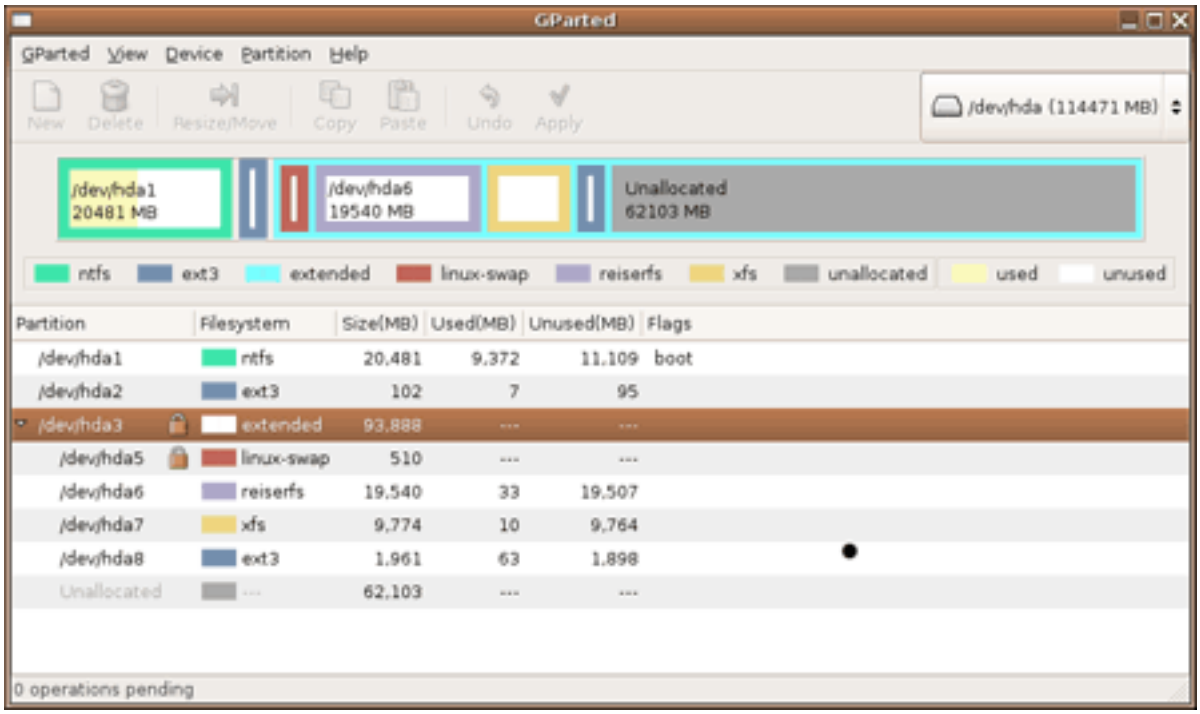


Another popular tool for working with the partition table is `parted`, which can resize and format many partition types as well as create and destroy them. While `parted` cannot resize NTFS partitions, `ntfsresize` can. The `qtparted` tool uses the Qt toolkit to provide a graphical interface. It includes the `parted` functions as well as `ntfsresize` functions.

The `gparted` tool is another graphical partitioning tool, designed for the GNOME desktop. It uses the GTK+GUI library, and is shown in Figure 2. (See Resources for

links to both `qtparted` and `gparted`.)

Figure 2. Using gparted



Several commercial partitioning tools are available. Perhaps the best known one is PartitionMagic, now sold by Symantec.

Many distributions allow you to partition your disk, and sometimes shrink an existing Windows NTFS or FAT32 partition, as part of the installation process. Consult the installation documentation for your distribution.

Logical volume manager

The logical volume manager (or LVM) for Linux allows you to combine multiple physical storage devices into a single *volume group*. For example, you might add a partition to an existing volume group, rather than having to carve out contiguous space large enough for your desired filesystem.

RAID

RAID (Redundant Array of Independent Disks) is a technology for providing a reliable data storage using low-cost disks that are much less expensive than the disks found on high-end systems. There are several different types of RAID, and RAID may be implemented in hardware or software. Linux supports both hardware and software RAID.

More filesystems

You will probably encounter filesystems besides those discussed above.

IBM's *Journaled File System (JFS)*, currently used in IBM enterprise servers, is

designed for high-throughput server environments. It is available for Linux and is included in several distributions. To create JFS filesystems, use the `mkfs.jfs` command.

There are other filesystems too, such as the cramfs filesystem often used on embedded devices.

The next section shows you how to maintain integrity on filesystems and what to do when things go wrong.

Section 3. Filesystem integrity

This section covers material for topic 1.104.2 for the Junior Level Administration (LPIC-1) exam 101. The topic has a weight of 3.

In this section, you learn how to:

- Monitor free space and inodes
- Verify the integrity of filesystems
- Repair simple filesystem problems

Both standard and journaling filesystems are covered. The emphasis is on ext2 and ext3, but the tools for other filesystems are mentioned as well. Most of this material applies to both 2.4 and 2.6 kernels. The examples in this section mostly use Ubuntu 5.10 "Breezy Badger" (a distribution based on Debian), with a 2.6.12 kernel, which was installed on the filesystems we created in the previous section. Your results on other systems are likely to differ.

Monitoring free space

First, a review. In the tutorial for topic 103, "LPI exam 101 prep: GNU and UNIX commands," you learned that a file or directory is contained in a collection of *blocks*, and information about the file or directory is contained in an *inode*.

Both the data blocks and the inode blocks take space on filesystems, so you need to monitor the space usage to ensure that your filesystems have space for growth.

df

`df` displays information about mounted filesystems. (You will learn more about mounting filesystems in the next section, Mounting and unmounting filesystems). If you add the `-T` option, then the filesystem type is included in the display; otherwise, it is not. The output from `df` for the Ubuntu system that we installed on the

filesystems created in the previous section is shown in Listing 17.

Listing 17. Displaying filesystem usage

```
ian@pinguino:~$ df -T
Filesystem    Type   1K-blocks      Used Available Use% Mounted on
/dev/hda6 reiserfs    20008280   1573976  18434304   8% /
tmpfs        tmpfs     1034188         0   1034188   0% /dev/shm
tmpfs        tmpfs     1034188     12588   1021600   2% /lib/modules/2.6.12-10-386/volatile
/dev/hda2     ext3      101105     19173     76711  20% /boot
/dev/hda8     vfat     2004156         8   2004148   1% /dos
/dev/hda7      xfs     9998208      3544   9994664   1% /home
/dev/hda1     ntfs    20967416   9594424  11372992  46% /media/hda1
```

You will notice that the output includes the total number of blocks as well as the number used and free. You will also notice the filesystem, such as /dev/hda7, and its mount point: /home for /dev/hda7. The two *tmpfs* entries are for virtual memory filesystems. These exist only in RAM or swap space and are created when mounted without need for a `mkfs` command. You can read more about tmpfs in "Common threads: Advanced filesystem implementor's guide, Part 3" (see Resources for a link).

If you want specific information on inode usage, use the `-i` option on the `df` command. You can exclude certain filesystem types using the `-x` option or restrict information to just certain filesystem types using the `-t` option. Use these multiple time if necessary. See the examples in Listing 18.

Listing 18. Displaying inode usage

```
ian@pinguino:~$ df -i -x tmpfs
Filesystem            Inodes   IUsed   IFree IUse% Mounted on
/dev/hda6                  0       0       0    -  /
/dev/hda2              26208      34   26174    1% /boot
/dev/hda8                  0       0       0    -  /dos
/dev/hda7            10008448     176 10008272    1% /home
/dev/hda1               37532   36313    1219   97% /media/hda1
ian@pinguino:~$ df -iT -t vfat -t ext3
Filesystem    Type    Inodes   IUsed   IFree IUse% Mounted on
/dev/hda2     ext3     26208      34   26174    1% /boot
/dev/hda8     vfat         0       0       0    -  /dos
```

Perhaps you are not surprised to see that the FAT32 filesystem does not have inodes, but it may surprise you to see that the ReiserFS information shows no inodes. ReiserFS keeps metadata for files and directories in *stat items*. And since ReiserFS uses a balanced tree structure, there is no predetermined number of inodes as there are, for example, in ext2, ext3, or xfs filesystems.

There are several other options you may use with `df` to limit the display to local filesystems or control the format of output. For example, use the `-H` option to display human readable sizes, such as 1K for 1024, or use the `-h` (or `--si`) option to get sizes in powers of 10 (1K=1000).

If you aren't sure which filesystem a particular part of your directory tree lives on, you can give the `df` command a parameter of a directory name or even a filename as shown in Listing 19.

Listing 19. Human readable output for df

```
ian@pinguino:~$ df --si ~ian/index.html
Filesystem             Size   Used  Avail Use% Mounted on
/dev/hda7               11G   3.7M    11G   1% /home
```

du

The `df` command gives information about only a whole filesystem. Sometimes you might want to know how much space is used by your home directory, or how big a partition you would need if you wanted to move /usr to its own filesystem. To answer this kind of question, use the `du`command.

The `du` command displays information about the filename (or filenames) given as parameters. If a directory name is given, then `du` recurses and calculates sizes for every file and subdirectory of the given directory. The result can be a lot of output. Fortunately, you can use the `-s` option to request just a summary for a directory. If you use `du` to get information for multiple directories, then add the `-c` option to get a grand total. You can also control output format with the same set of size options (`-h`, `-H`, `--si`, and so on) that are used for `df`. Listing 20 shows two views of my home directory on the newly installed Ubuntu system.

Listing 20. Using du

```
ian@pinguino:~$ du -hc *
0       Desktop
16K     index.html
16K     total
ian@pinguino:~$ du -hs .
3.0M    .
```

The reason for the difference between the 16K total from `du -c *` and the 3M summary from `du -s` is that the latter includes the entries starting with a dot, such as .bashrc, while the former does not.

One other thing to note about `du` is that you must be able to read the directories that you are running it against.

So now, let's use `du` to display the total space used by the /usr tree and each of its first-level subdirectories. The result is shown in Listing 21. Use root authority to make sure you have appropriate access permissions.

Listing 21. Using du on /usr

```
root@pinguino:~# du -shc /usr/*
66M     /usr/bin
0       /usr/doc
1.3M    /usr/games
742K    /usr/include
0       /usr/info
497M    /usr/lib
0       /usr/local
7.3M    /usr/sbin
```

```
578M    /usr/share
0       /usr/src
14M     /usr/X11R6
1.2G    total
```

Checking filesystems

Sometimes your system may crash or lose power. In these cases, Linux will not be able to cleanly unmount your filesystems, and they may be left in an inconsistent state, with some changes completed and some not. Operating with a damaged filesystem is not a good idea as you are likely to further compound any existing errors.

The main tool for checking filesystems is `fsck`, which, like `mkfs`, is really a front end to filesystem checking routines for the various filesystem types. Some of the underlying check routines are shown in Listing 22.

Listing 22. Some of the fsck programs

```
ian@pinguino:~$ ls /sbin/*fsck*
/sbin/dosfsck      /sbin/fsck.ext3   /sbin/fsck.reiser4   /sbin/jfs_fscklog
/sbin/e2fsck       /sbin/fsck.jfs    /sbin/fsck.reiserfs  /sbin/reiserfsck
/sbin/fsck         /sbin/fsck.minix  /sbin/fsck.vfat
/sbin/fsck.cramfs  /sbin/fsck.msdos  /sbin/fsck.xfs
/sbin/fsck.ext2    /sbin/fsck.nfs    /sbin/jfs_fsck
```

The system boot process use `fsck` to check the root filesystem and any other filesystems that are specified in the /etc/fstab control file. If the filesystem was not cleanly unmounted, a consistency check is performed. This is controlled by the *pass* (or *passno*) field (the sixth field) of the /etc/fstab entry. Filesystems with pass set to zero are not checked at boot time. The root filesystem has a pass value of 1 and is checked first. Other filesystems will usually have a pass value of 2 (or higher), indicating the order in which they should be checked. Multiple `fsck` operations can run in parallel, so different filesystems are allowed to have the same pass value, as is the case for our example /boot and /home filesystems.

Listing 23. Boot checking of filesystems with fstab entries

```
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
proc            /proc           proc    defaults        0       0
/dev/hda6       /               reiserfs defaults       0       1
/dev/hda2       /boot           ext3    defaults        0       2
/dev/hda8       /dos            vfat    defaults        0       0
/dev/hda7       /home           xfs     defaults        0       2
```

Note that some journaling filesystems, such as ReiserFS and xfs, might have a pass value of 0 because the journaling code, rather than `fsck`, does the filesystem consistency check and repair.

Repairing filesystems

If the automatic boot time check of filesystems is unable to restore consistency, you are usually dumped into a single user shell with some instructions to run `fsck` manually. For an ext2 filesystem, which is not journaled, you may be presented with a series of requests asking you to confirm proposed actions to fix particular blocks on the filesystem. You should generally allow `fsck` to attempt to fix problems, by responding `y` (for yes). When the system reboots, check for any missing data or files.

If you suspect corruption, or want to run a check manually, most of the checking programs require the filesystem to be unmounted first. Since you can't unmount the root filesystem on a running system, the best you can do is drop to single user mode (using `telinit 1`) and then remount the root filesystem read-only, at which time you should be able to perform a consistency check. (Mounting filesystems is covered in the next section; Mounting and unmounting filesystems.) A better way to check a filesystem is to boot a recovery system, such as a live CD or a USB memory key, and perform the check of your unmounted filesystems from that.

Why journal?

An `fsck` scan of an ext2 disk can take quite a while to complete, because the internal data structure (or *metadata*) of the filesystem must be scanned completely. As filesystems get larger and larger, this takes longer and longer, even though disks also keep getting faster, so a full check may take one or more hours.

This problem was the impetus for *journaled* or *journaling* filesystems. Journaled filesystems keep a log of recent changes to the filesystem metadata. After a crash, the filesystem driver inspects the log in order to determine which recently changed parts of the filesystem may possibly have errors. With this design change, checking a journaled filesystem for consistency typically takes just a matter of seconds, regardless of filesystem size. Furthermore, the filesystem driver will usually check the filesystem on mounting, so an external `fsck` check is generally not required. In fact, for the xfs filesystem, `fsck` does nothing!

If you do run a manual check of a filesystem, check the man pages for the appropriate `fsck` command (`fsck.ext3`, `e2fsck` , `reiserfsck`, and so on) to determine the appropriate parameters. Some examples are in Listing 24, using a Ubuntu live CD image to run the `fsck` commands.

Listing 24. Running fsck manually

```
root@ubuntu:~# fsck -p /dev/hda6
fsck 1.38 (30-Jun-2005)
Reiserfs super block in block 16 on 0x306 of format 3.6 with standard journal
Blocks (total/free): 5002224/4608416 by 4096 bytes
Filesystem is clean
Replaying journal..
Reiserfs journal '/dev/hda6' in blocks [18..8211]: 0 transactions replayed
Checking internal tree..finished
root@ubuntu:~# fsck -p /dev/hda2
fsck 1.38 (30-Jun-2005)
BOOT: clean, 34/26208 files, 22488/104420 blocks
root@ubuntu:~# fsck -p /dev/hda7
fsck 1.38 (30-Jun-2005)
root@ubuntu:~# fsck -a /dev/hda8
```

```
fsck 1.38 (30-Jun-2005)
dosfsck 2.11, 12 Mar 2005, FAT32, LFN
/dev/hda8: 1 files, 2/501039 clusters
```

Advanced tools

There are several more advanced tools that you can use to examine or repair a filesystem. Check the man pages for the correct usage and the Linux Documentation Project (see Resources) for how-to information. Almost all of these commands require a filesystem to be unmounted, although some functions can be used on filesystems that are mounted read-only. A few of the commands are described below.

You should always back up your filesystem before attempting any repairs.

Tools for ext2 and ext3 filesystems

tune2fs
Adjusts parameters on ext2 and ext3 filesystems. Use this to add a journal to an ext2 system, making it an ext3 system, as well as display or set the maximum number of mounts before a check is forced. You can also assign a label and set or disable various optional features.

dumpe2fs
Prints the super block and block group descriptor information for an ext2 or ext3 filesystem.

debugfs
Is an interactive file system debugger. Use it to examine or change the state of an ext2 or ext3file system.

Tools for Reiserfs filesystems

reiserfstune
Displays and adjusts parameters on ReiserFS filesystems.

debugreiserfs
Performs similar functions to dumpe2fs and debugfs for ReiserFS filesystems.

Tools for XFS filesystems

xfs_info
Displays XFS filesystem information.

xfs_growfs
Expands an XFS filesystem (assuming another partition is available).

xfs_admin
Changes the parameters of an XFS filesystem.

xfs_repair
Repairs an XFS filesystem when the mount checks are not sufficient to repair the system.

xfs_db
Examines or debugs an XFS filesystem.

Section 4. Mounting and unmounting filesystems

This section covers material for topic 1.104.3 for the Junior Level Administration (LPIC-1) exam 101. The topic has a weight of 3.

In this section, you learn how to:

- Mount filesystems
- Unmount filesystems
- Configure filesystem mounting on bootup
- Configure user-mountable, removable filesystems such as tape drives, floppies, and CDs

Mounting filesystems

The Linux filesystem is one big tree rooted at /, and yet we have filesystems on different devices and partitions. Now we resolve this apparent incongruity. The root (/) filesystem is mounted as part of the initialization process. Each of the other filesystems that we have created is not usable by your Linux system until it is *mounted* at a *mount point*.

A mount point is simply a directory in the current set of mounted filesystems at which point the filesystem on a device is grafted into the tree. Mounting is the process of making the filesystem on the device part of your accessible Linux filesystem. For example, you might mount filesystems on hard drive partitions as /boot, /tmp, or /home, and you might mount the filesystem on a floppy drive as /mnt/floppy and the filesystem on a CD-ROM as /media/cdrom1.

Besides filesystems on partitions, floppy disks, and CDs, there are other types of filesystems. We alluded briefly to the tmpfs filesystem, which is a virtual memory filesystem. It is also possible to mount filesystems from one system on another system using a networked filesystem such as NFS or AFS. You can also create a file in an existing filesystem and format that as a, possibly different, kind of filesystem and mount that too.

While the mount process actually mounts the *filesystem* on some device (or other

resource), it is common to simply say that you "mount the device", which is understood to mean "mount the filesystem on the device".

The basic form of the `mount` command takes two parameters: the device (or other resource) containing the filesystem to be mounted, and the mount point. For example, we mount our FAT32 partition /dev/hda8 at the mount point /dos as shown in Listing 25.

Listing 25. Mounting /dos

```
root@pinguino:~# mount /dev/hda8 /dos
```

The mount point must exist before you mount anything over it. When you do, the files on the filesystem you are mounting become the files and subdirectories of the mount point. If the mount point directory already contained files or subdirectories, they are no longer visible until the mounted filesystem is unmounted, at which point they become visible again. It is a good idea to avoid this problem by using only empty directories as mount points.

After mounting a filesystem, any files or directories created or copied to the mount point or any directory below it will be created on the mounted filesystem. So a file such as /dos/sampdir/file.txt will be created on the FAT32 filesystem that we mounted at /dos in our example.

Usually, the `mount` command will automatically detect the type of filesystem being mounted. Occasionally you may need to specify the filesystem type explicitly using the `-t` option as shown in Listing 26.

Listing 26. Mounting with explicit filesystem type

```
root@pinguino:~# mount -t vfat /dev/hda8 /dos
```

To see what filesystems are mounted, use the `mount` command with no parameters. Listing 27 shows our example system.

Listing 27. Displaying mounted filesystems

```
/dev/hda6 on / type reiserfs (rw)
proc on /proc type proc (rw)
sysfs on /sys type sysfs (rw)
devpts on /dev/pts type devpts (rw,gid=5,mode=620)
tmpfs on /dev/shm type tmpfs (rw)
usbfs on /proc/bus/usb type usbfs (rw)
tmpfs on /lib/modules/2.6.12-10-386/volatile type tmpfs (rw,mode=0755)
/dev/hda2 on /boot type ext3 (rw)
/dev/hda8 on /dos type vfat (rw)
/dev/hda7 on /home type xfs (rw)
/dev/hda1 on /media/hda1 type ntfs (rw)
tmpfs on /dev type tmpfs (rw,size=10M,mode=0755)
```

You can also view similar information by displaying /proc/mounts or /etc/mtab, both of which contain information about mounted filesystems.

Mount options

The `mount` command has several options that override the default behavior. For example, you can mount a filesystem read-only by specifying `-o ro`. If the filesystem is already mounted, add `remount` as shown in Listing 28.

Listing 28. Remounting read-only

```
root@pinguino:~# mount -o remount,ro /dos
```

Notes:

- Separate multiple options with commas.
- When remounting an already mounted filesystem, it suffices to specify either the mount point or the device name. It is not necessary to specify both.
- You cannot mount a read-only filesystem as read-write. Media that cannot be modified, such as CD-ROM discs, will automatically be mounted read-only.
- To remount a writable device read-write, specify `-o remount,rw`

Remount commands will not complete successfully if any process has open files or directories in the filesystem being remounted. Use the `lsof` command to determine what files are open. Check the man pages to learn about additional mount options and `lsof`.

fstab

In the tutorial for topic 102, "LPI exam 101 prep (topic 102)d: Linux installation and package management," you learned how to use the `root=` parameter in both GRUB and LILO to tell the boot loader what filesystem should be mounted as root. Once this filesystem is mounted, the initialization process runs `mount` with the `-a` option to automatically mount a set of filesystems. The set is specified in the file /etc/fstab. Listing 29 shows /etc/fstab for the sample Ubuntu system that we installed using the filesystems created earlier in this tutorial.

Listing 29. An example fstab

```
root@pinguino:~# cat /etc/fstab
# /etc/fstab: sttic file system information.
#
#<file system> <mount point>   <type>  <options>       <dump>  <pass>
proc            /proc           proc    defaults        0       0
/dev/hda6       /               reiserfs defaults       0       1
/dev/hda2       /boot           ext3    defaults        0       2
/dev/hda8       /dos            vfat    defaults        0       0
/dev/hda7       /home           xfs     defaults        0       2
```

```
/dev/hda1       /media/hda1     ntfs    defaults        0       0
/dev/hda5       none            swap    sw              0       0
/dev/hdc        /media/cdrom0   udf,iso9660 user,noauto     0       0
/dev/fd0        /media/floppy0  auto    rw,user,noauto  0       0
```

Lines starting with a # character are comments. Remaining lines contain six fields. Since the fields are positional, they must all be specified.

file system

For the examples used so far, this will be a device name such as /dev/hda1.

mount point

This is the mount point we discussed in Mounting filesystems above. For swap space, this should be the value `none`. For ext2, ext3, and xfs filesystems, you may also specify a label such as `LABEL=XFSHOME`. This makes your system more robust when devices are added or removed.

type

Specifies the type of filesystem. CD/DVD drives will often support either ISO9660 or UDF filesystems, so you may specify multiple possibilities in a comma-separated list. If you want `mount` to automatically determine the type, specify `auto` as is done in the last line for the floppy drive.

option

Specifies the mount options. Specify `defaults` if you want default mount options. Some options you will want to know about are:

- `rw` and `ro` specify whether the filesystem should be mounted read-write or read-only.
- `noauto` specifies that this filesystem should not be automatically mounted at boot time or whenever `mount -a` is issued. In our example, this is done for the removable drives.
- user
- Specifies that a non-root user is permitted to mount and unmount the filesystem. This is especially useful for removable media. This option must be specified in /etc/fstab rather than on the `mount` command.
- `exec` and `noexec` specify whether or not to allow execution of files from the mounted filesystem. User-mounted filesystems default to `noexec` unless `exec` is specified **after** `user`.
- `noatime` will disable recording of access times. Not using access times may improve performance.

dump

Specifies whether the `dump` command should consider this ext2 or ext3 filesystem for backups. A value of 0 tells `dump` to ignore this filesystem.

pass

Non-zero values of pass specify the order of checking filesystems at boot time,

as discussed in Checking filesystems.

For filesystems that are listed in /etc/fstab, it suffices to give either the device name or the mount point when mounting the filesystem. You do not need to give both.

Consult the man pages for `fstab` and `mount` for additional information, including options not covered here.

Unmounting filesystems

All mounted filesystems are usually unmounted automatically by the system when it is rebooted or shut down. When a filesystem is unmounted, any cached filesystem data in memory is flushed to the disk.

You may also unmount filesystems manually. Indeed, you **should** do this when removing writable media such as diskettes or USB drives or memory keys. Before unmounting a filesystem, make sure that there are no processes running that have open files on the filesystem. Then, use the `umount` command, specifying *either* the device name or mount point as an argument. Some successful and unsuccessful examples are shown in Listing 30.

Listing 30. Unmounting filesystems

```
root@pinguino:~# lsof /dos
root@pinguino:~# umount /dos
root@pinguino:~# mount /dos
root@pinguino:~# umount /dev/hda8
root@pinguino:~# umount /boot
umount: /boot: device is busy
umount: /boot: device is busy
root@pinguino:~# lsof /boot
COMMAND  PID USER   FD   TYPE DEVICE   SIZE NODE NAME
klogd   6498 klog    1r   REG    3,2 897419 6052 /boot/System.map-2.6.12-10-386
```

After a filesystem is unmounted, any files in the directory used for the mount point are visible again.

Swap space

You may have noticed in the discussion of `fstab` above that swap space does not have a mount point. The boot process usually enables swap space defined in /etc/fstab unless the `noauto` option is specified. To manually control swap space on a running system -- for example, if you added a new swap partition -- use the `swapon` and `swapoff` commands. See the man pages for details.

You can view the currently enabled swap devices with `cat /proc/swaps`.

Section 5. Disk quotas

This section covers material for topic 1.104.4 for the Junior Level Administration (LPIC-1) exam 101. The topic has a weight of 3.

In this section, you learn how to:

- Enable quotas
- Set quota limits
- Check quotas
- Generate quota reports

Quotas allow you to control disk usage by user or by group. Quotas prevent individual users and groups from using a larger portion of a filesystem than they are permitted, or from filling it up altogether. Quotas must be enabled and managed by the root user. They are often used on multi-user systems, but less often on single-user workstations.

Enabling quotas

Quotas require kernel support. Generally, a recent 2.4 or 2.6 kernel should have all the support you need. Earlier versions may have incomplete quota support, requiring you to build your own kernel. In current implementations you will probably find quota support implemented as kernel modules. There are three different types of quota support; vfsold (version 1 quota), vfsv0 (version 2 quota), and xfs (quota on XFS filesystems). This section covers version 2 quota on non-XFS filesystems and xfs quota on XFS filesystems.

The first step to enable quotas is to add the `usrquota`, or `grpquota` options to the filesystem definitions in /etc/fstab, according to whether you want to implement user quotas, group quotas, or both. Suppose you want to add both types of quota to the XFS filesystem used for home directories in our example and also to the /boot filesystem so you can see how it works on two different filesystems. Do this as shown in Listing 31.

Listing 31. Enabling quota support in /etc/fstab

```
/dev/hda2       /boot           ext3    defaults,usrquota,grpquota    0   2
/dev/hda7       /home           xfs     defaults,usrquota,grpquota    0   2
```

For XFS filesystems, quota data is considered part of the filesystem metadata. For other filesystems, user quota information is stored in the aquota.user file in the root of the filesystem, and group quota is similarly stored in aquota.group. Version 1 quotas used quota.user and quota.group.

After you edit /etc/fstab and add quotas, you need to remount the filesystems and, for non-XFS filesystems, create the quota files and enable quota checking. The `quotacheck` command checks the quotas on all filesystems and creates the required aquota.user and aquota.group files if they do not exist. It can also repair damaged quota files. See the man pages for more information. The `quotaon` command turns on quota checking. Listing 32 shows an example. The following options are used on both commands:

-a
For all filesystems in /etc/fstab that are enabled for automount

-u
For user quotas (this is the default)

-g
For group quotas

-v
For verbose output

Listing 32. Creating quota files and turning quota on

```
root@pinguino:~# quotacheck -augv
quotacheck: Scanning /dev/hda2 [/boot] quotacheck: Cannot stat old user quota
file: No such file or directory
quotacheck: Cannot stat old group quota file: No such file or directory
quotacheck: Cannot stat old user quota file: No such file or directory
quotacheck: Cannot stat old group quota file: No such file or directory
done
quotacheck: Checked 4 directories and 23 files
quotacheck: Old file not found.
quotacheck: Old file not found.
quotacheck: Skipping /dev/hda7 [/home]
root@pinguino:~# quotaon -ugva
/dev/hda2 [/boot]: group quotas turned on
/dev/hda2 [/boot]: user quotas turned on
```

Checking quotas on boot

The `quotacheck` and `quotaon` commands are usually included in initialization scripts so that quotas are enabled whenever you reboot the system. The Quota Mini HOWTO (see Resources for a link) has additional information.

The `quotaoff` command disables quotas, should you ever need to do so.

Setting quota limits

As you have seen, quotas are controlled either through binary files in the root of the filesystem or through filesystem metadata. In order to set a quota for a particular user, use the `edquota` command. This command extracts the quota information for the user from the various filesystems with quotas enabled, creates a temporary file, and opens an editor for you to adjust the quotas. See the man pages for `edquota` to find out which editor is used. You must be root to edit quotas. The information

displayed will look something like Listing 33.

Listing 33. Running edquota

```
Disk quotas for user ian (uid 1000):
  Filesystem           blocks       soft       hard     inodes     soft     hard
  /dev/hda2                 0          0          0          0        0        0
  /dev/hda7              2948          0          0        172        0        0
```

As you can see, `edquota` displays my current usage of both 1K blocks and inodes on each of the filesystems that have quota turned on. There are also soft and hard limits for both block and inode usage. In this example, these are 0, meaning no quota limit is enforced.

The soft limit is the value at which a user will receive e-mail warnings about being over quota. The hard limit is the value that a user may not exceed. You can think of block limits as a limit on the amount of data that a user may store, and inode limits as a limit on the number of files and directories.

Changing quota limits

You change the quota limits by changing the values in the temporary file and then saving the file. Quit the file without saving if you do not want to make changes. Assume you want to set my quota to 10MB of data and 1000 files on the /home filesystem. Allowing 10% additional for hard limits, you would set values as in Listing 34.

Listing 34. Setting limits

```
Disk quotas for user ian (uid 1000):
  Filesystem           blocks       soft       hard     inodes     soft     hard
  /dev/hda2                 0          0          0          0        0        0
  /dev/hda7              2948      10240      11264        172     1000     1100
```

Save the file, and the new quotas will take effect. In this example, no changes were made to the quota for user ian on the /boot filesystem, since ian cannot write to this filesystem. Note also that any changes you make to the used blocks or inodes values will be ignored.

Copying quotas

Now suppose you are creating ids for people enrolled in a class. Assume you have users gretchen, tom, and greg and you'd like them all to have the same quota as ian. You do this using the `-p` option of `edquota`, which uses the quota values of ian as a prototype for those of the other users as shown in Listing 35.

Listing 35. Setting quotas from a prototype

```
root@pinguino:~# edquota -p ian gretchen tom greg
```

Group limits

You can also use `edquota` to restrict the allocation of disk space based on the group ownership of files. For example, the three class attendees above are set up in primary group xml-101. To limit the total amounts used by the all members of the group to 25MB and 2500 files, use the command `edquota -q xml-101` and set the values as shown in Listing 36.

Listing 36. Setting quotas for a group

```
Disk quotas for group xml-101 (gid 1001):
  Filesystem           blocks       soft       hard     inodes     soft     hard
  /dev/hda2                 0          0          0          0        0        0
  /dev/hda7                28      25600      28160         10     2500     2750
```

The grace period

Users may exceed their soft limit for a *grace period*, which defaults to 7 days. After the grace period, the soft limit is enforced as a hard limit. Set grace periods with the `-y` option of `edquota`. Again, you will be placed in an editor with data similar to that of Listing 37. As before, save changes to update the values. Be sure to leave your users enough time to receive their warning e-mail and delete some files.

Listing 37. Setting grace periods

```
Grace period before enforcing soft limits for users:
Time units may be: days, hours, minutes, or seconds
  Filesystem             Block grace period     Inode grace period
  /dev/hda2                     7days                  7days
  /dev/hda7                     7days                  7days
```

Checking quotas

The `quota` command with no options displays the quotas for the invoking user on any filesystems for which the user has quotas set. The `-v` option displays the information for all filesystems that have quota enabled. The root user may also add a user name to the command to view quotas for a particular user. These commands are shown in Listing 38.

Listing 38. Displaying quotas

```
root@pinguino:~# quota
Disk quotas for user root (uid 0): none
root@pinguino:~# quota -v
Disk quotas for user root (uid 0):
     Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
      /dev/hda2   19173       0       0              26       0       0
      /dev/hda7      16       0       0               5       0       0
root@pinguino:~# quota -v ian
Disk quotas for user ian (uid 1000):
     Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
      /dev/hda2       0       0       0               0       0       0
      /dev/hda7    2948   10240   11264             172    1000    1100
```

Along with the statistics on current usage, you see the soft and hard quota limits displayed. Listing 39 shows what happens if you exceed the soft limit and then what happens if you attempt to exceed the hard limit. In this example, a file of approximately 4MB is created and then a copy is made. Along with the original usage of approximately 3MB, this is sufficient to exceed the soft limit. Notice how the soft limit has an asterisk beside it indicating that the user is over quota. Note also that the grace period columns now indicate how long the user has to correct the problem.

Listing 39. Exceeding quotas

```
ian@pinguino:~$ dd if=/dev/zero of=big1 bs=512 count=8000
8000+0 records in
8000+0 records out
4096000 bytes transferred in 0.019915 seconds (205674545 bytes/sec)
ian@pinguino:~$ cp big1 big2
ian@pinguino:~$ quota
Disk quotas for user ian (uid 1000):
     Filesystem  blocks   quota   limit   grace   files   quota   limit   grace
      /dev/hda7   10948*  10240   11264   7days     174    1000    1100
ian@pinguino:~$ cp big1 big3
cp: writing `big3': Disk quota exceeded
```

Generating quota reports

Checking user quotas one user at a time is not very useful, so you will want to use the `repquota` command to generate quota reports. Listing 40 shows how to see the quotas for all users and groups on /home.

Listing 40. Exceeding quotas

```
root@pinguino:~# repquota -ug /home
*** Report for user quotas on device /dev/hda7
Block grace time: 7days; Inode grace time: 7days
                        Block limits                File limits
User            used    soft    hard  grace    used  soft  hard  grace
----------------------------------------------------------------------
root      --      16       0       0              5     0     0
ian       +-   11204   10240   11264  6days     175  1000  1100
tom       --       8   10240   11264              3  1000  1100
gretchen  --       8   10240   11264              3  1000  1100
greg      --      12   10240   11264              4  1000  1100


*** Report for group quotas on device /dev/hda7
Block grace time: 7days; Inode grace time: 7days
                        Block limits                File limits
Group           used    soft    hard  grace    used  soft  hard  grace
----------------------------------------------------------------------
root      --      16       0       0              5     0     0
ian       --   11204       0       0            175     0     0
xml-101   --      28   25600   28160             10  2500  2750
```

Note the plus sign in the listing for user ian, indicating that ian is over quota.

As with other quota commands, the `-a` option produces a report for all mounted

filesystems that have quota enabled. The `-v` option produces more verbose output. And the `-n` option produces the listing by numeric user number rather than resolving the user number to a name. This may provide a performance boost for large reports, but is generally less useful to human readers.

Warning users

The `warnquota` command is used to send e-mail warnings to users who are over quota. When a group is over quota, the e-mail is sent to the user specified in /etc/quotagrpadmins for the group. Normally `warnquota` is run periodically as a `cron` job. See the man pages for `cron` and `warnquota` for more information.

Section 6. File permissions and access control

This section covers material for topic 1.104.5 for the Junior Level Administration (LPIC-1) exam 101. The topic has a weight of 5.

In this section, you learn about:

- Users and groups
- Permissions on files and directories
- Changing permissions
- Access modes
- Immutable files
- Default file creation modes

User and groups

By now, you know that Linux is a multi-user system and that each user belongs to one *primary* group and possibly additional groups. It is also possible to log in as one user and become another user using the `su` or `sudo -s` commands. Ownership of files in Linux is closely related to user ids and groups, so let's review some basic user and group information.

Who am I?

If you have not become another user, your id is still the one you used to log in. If you have become another user, your prompt may include your user id, as most of the examples in this tutorial do. If your prompt does not include your user id, then you can use the `whoami` command to check your current effective id. Listing 41 shows

some examples where the prompt strings (from the PS1 environment variable) are different from the other examples in this tutorial.

Listing 41. Determining effective user id

```
/home/ian$ whoami
tom
/home/ian$ exit
exit
$ whoami
ian
```

What groups am I in?

Similarly, you can find out what groups you are in by using the `groups` command. You can find out both user and group information using the `id` command. Add a user id parameter to either `groups` or `id` to see information for that user id instead of the current user id. See Listing 42 for some examples.

Listing 42. Determining group membership

```
$ su tom
Password:
/home/ian$ groups
xml-101
/home/ian$ id
uid=1001(tom) gid=1001(xml-101) groups=1001(xml-101)
/home/ian$ exit
$ groups
ian adm dialout cdrom floppy audio dip video plugdev lpadmin scanner admin xml-101
$ id
uid=1000(ian) gid=1000(ian) groups=4(adm),20(dialout),24(cdrom),25(floppy),
29(audio),30(dip),44(video),46(plugdev),104(lpadmin),105(scanner),106(admin),
1000(ian),1001(xml-101)
$ groups tom
tom : xml-101
```

File ownership and permissions

Just as every user has an id and is a member of one primary group, so every file on a Linux system has one owner and one group associated with it.

Ordinary files

Use the `ls -l` command to display the owner and group.

Listing 43. Determining file ownership

```
gretchen@pinguino:~$ ls -l /bin/bash .bashrc
-rw-r--r--  1 gretchen xml-101   2227 Dec 20 10:06 .bashrc
-rwxr-xr-x  1 root     root    645140 Oct  5 08:16 /bin/bash
```

In this particular example, user gretchen's .bashrc file is owned by her and is in the xml-101 group, which is her primary group. Similarly, /bin/bash is owned by user root

and is in the group root. User names and groups names come from separate namespaces, so a group name may be the same as a user name. In fact, many distributions default to creating a matching group for each new user.

The Linux permissions model has three types of permission for each filesystem object. The permissions are read (r), write (w), and execute (x). Write permission includes the ability to alter or delete an object. In addition, these permissions are specified separately for the file's owner, members of the file's group, and everyone else.

Referring back to the first column of Listing 43, notice that it contains a ten-character string. The first character describes the type of object (- for an ordinary file in this example) and the remaining nine characters represent three groups of three characters. The first group indicates the read, write, and execute permissions for the file's owner. A - indicates that the corresponding permission is not granted. So user gretchen can read and write the .bashrc file, but not execute it; while root can read, write, **and** execute the /bin/bash file. The second group indicates the read, write, and execute permissions for the file's group. Members of the xml-101 group can read gretchen's .bashrc file, but not write it, as can everyone else. Similarly, members of the root group and everyone else can read or execute the /bin/bash file.

Directories

Directories use the same permissions flags as regular files but they are interpreted differently. Read permission for a directory allows a user with that permission to list the contents of the directory. Write permission means a user with that permission can create or delete files in the directory. Execute permission allows the user to enter the directory and access any subdirectories. Without execute permission, the filesystem objects inside a directory are not accessible. Without read permission, the filesystem objects inside a directory are not viewable, but these objects can still be accessed as long as you know the full path to the object on disk. Listing 44 is a somewhat artificial example that illustrates these points.

Listing 44. Permissions and directories

```
ian@pinguino:~$ ls -l /home
total 8
drwxr-x---   2 greg     xml-101   60 2005-12-20 11:37 greg
drwx------  13 gretchen xml-101 4096 2005-12-21 12:22 gretchen
drwxr-xr-x  15 ian      ian     4096 2005-12-21 10:25 ian
d-wx--x--x   2 tom      xml-101   75 2005-12-21 11:05 tom
ian@pinguino:~$ ls -a ~greg
.  ..  .bash_history  .bash_profile  .bashrc
ian@pinguino:~$ ls -a ~gretchen
ls: /home/gretchen: Permission denied
ian@pinguino:~$ ls -a ~tom
ls: /home/tom: Permission denied
ian@pinguino:~$ head -n 3 ~tom/.bashrc
# ~/.bashrc: executed by bash(1) for non-login shells.
# see /usr/share/doc/bash/examples/startup-files (in the package bash-doc)
# for examples
```

The first character of a long listing describes the type of object (d for a directory). User greg's home directory has read and execute permission for members of the

xml-101 group, so user ian can list the directory. User gretchen's home directory has neither read nor execute permission, so user ian cannot access it. User's tom's home has execute but not read permission, so user ian cannot list the contents, but can access objects within the directory if he knows they exist.

Other filesystem objects

A long listing may contain filesystem objects other than files and directories as shown by the first character in the listing. We will see more of these later in this section, but for now, note the possible types of object.

Table 3. Filesystem object types

Code	Object type
-	Regular file
d	Directory
l	Symbolic link
c	Character special device
b	Block special device
p	FIFO
s	Socket

Changing permissions

Adding permissions

Suppose you create a "Hello world" shell script. When you first create the script, it will usually not be executable. Use the `chmod` command with the `+x` option to add the execute permissions as shown in Listing 45.

Listing 45. Creating an executable shell script

```
ian@pinguino:~$ echo 'echo "Hello world!"'>hello.sh
ian@pinguino:~$ ls -l hello.sh
-rw-r--r--  1 ian ian 20 2005-12-22 12:57 hello.sh
ian@pinguino:~$ ./hello.sh
-bash: ./hello.sh: Permission denied
ian@pinguino:~$ chmod +x hello.sh
ian@pinguino:~$ ./hello.sh
Hello world!
ian@pinguino:~$ ls -l hello.sh
-rwxr-xr-x  1 ian ian 20 2005-12-22 12:57 hello.sh
```

You can use `r` to set the read permissions, and `w` to set the write permissions in a similar manner. In fact, you can use any combination of `r`, `w`, and `x` together. For example, using `chmod +rwx` would set all the read, write, and execute permissions for a file. This form of `chmod` adds permissions that are not already set.

Being selective

You may have noticed in the above example that execute permission was set for the owner, group, **and** others. To be more selective, you may prefix the mode expression with u to set the permission for users, g to set it for groups, and o to set it for others. Specifying a sets the permission for all users, which is equivalent to omitting it. Listing 46 shows how to add user and group write and execute permissions to another copy of the shell script.

Listing 46. Selectively adding permissions

```
ian@pinguino:~$ echo 'echo "Hello world!"'>hello2.sh
ian@pinguino:~$ chmod ug+xw hello2.sh
ian@pinguino:~$ ls -l hello2.sh
-rwxrwxr--  1 ian ian 20 2005-12-22 13:17 hello2.sh
```

Removing permissions

Sometimes you need to remove permissions rather than add them. Simply change the + to a `-`, and you remove any of the specified permissions that are set. Listing 47 shows how to remove all permissions for other users on the two shell scripts.

Listing 47. Removing permissions

```
ian@pinguino:~$ ls -l hello*
-rwxrwxr--  1 ian ian 20 2005-12-22 13:17 hello2.sh
-rwxr-xr-x  1 ian ian 20 2005-12-22 12:57 hello.sh
ian@pinguino:~$ chmod o-xrw hello*
ian@pinguino:~$ ls -l hello*
-rwxrwx---  1 ian ian 20 2005-12-22 13:17 hello2.sh
-rwxr-x---  1 ian ian 20 2005-12-22 12:57 hello.sh
```

Note that you can change permissions on more than one file at a time. As with some other commands you met in the tutorial for topic 103, you can even use the `-R` (or `--recursive`) option to operate recursively on directories and files.

Setting permissions

Now that you can add or remove permissions, you may wonder how to set just a specific set of permissions. Do this using = instead of + or `-`. To set the permissions on the above scripts so that other users have no access rights, you could use `chmod o= hello*`, instead of the command we used to remove permissions.

If you want to set different permissions for user, group, or other, you can separate different expressions by commas; for example, `ug=rwx,o=rx`, or you can use numeric permissions, which are described next.

Octal permissions

So far you have used symbols (ugoa and rxw) to specify permissions. There are three possible permissions in each group. You can also set permissions using octal numbers instead of symbols. Permissions set in this way use up to four octal digits.

We will look at the first digit when we discuss attributes. The second digit defines user permissions, the third group permissions and the fourth other permissions. Each of these three digits is constructed by adding the desired permissions settings: read (4), write (2), and execute (1). In the example for hello.sh in Listing 45, the script was created with permissions -rw-r--r--, corresponding to octal 644. Setting execute permission for everyone changed the mode to 755.

Using numeric permissions is very handy when you want to set all the permissions at once without giving the same permissions to each of the groups. Use Table 4 as a handy reference for octal permissions.

Table 4. Numeric permissions

Symbolic	Octal
`rwx`	7
`rw-`	6
`r-x`	5
`r--`	4
`-wx`	3
`-w-`	2
`--x`	1
`---`	0

Access modes

When you log in, the new shell process runs with your user and group ids. These are the permissions that govern your access to any files on the system. This usually means that you cannot access files belonging to others and cannot access system files. In fact, we as users are totally dependent on other programs to perform operations on our behalf. Because the programs you start inherit *your* user id, they cannot access any filesystem objects for which you haven't been granted access.

An important example is the /etc/passwd file, which cannot be changed by normal users directly, because write permission is enabled only for root: However, normal users need to be able to modify /etc/passwd somehow, whenever they need to change their password. So, if the user is unable to modify this file, how can this be done?

suid and sgid

The Linux permissions model has two special access modes called suid (set user id) and sgid (set group id). When an executable program has the suid access modes set, it will run as if it had been started by the file's owner, rather than by the user who really started it. Similarly, with the sgid access modes set, the program will run as if the initiating user belonged to the file's group rather than to his own group. Either or both access modes may be set.

Listing 48 shows that the `passwd` executable is owned by root:

Listing 48. suid access mode on /usr/bin/passwd

```
ian@pinguino:~$ ls -l /usr/bin/passwd
-rwsr-xr-x  1 root root 25648 2005-10-11 12:14 /usr/bin/passwd
```

Note that in place of an `x` in the user's permission triplet, there's an `s`. This indicates that, for this particular program, the suid and executable bits are set. So when `passwd` runs, it will execute as if the root user had launched it , with full superuser access, rather than that of the user who ran it. Because `passwd` runs with `root` access, it can modify /etc/passwd.

The suid and sgid bits occupy the same space as the `x` bits in a long directory listing. If the file is executable, the suid or sgid bits, if set, will be displayed as lowercase `s`, otherwise they are displayed as uppercase `S`.

While suid and sgid are handy, and even necessary in many circumstances, improper use of these access mode can allow breaches of a system's security. You should have as few suid programs as possible. The `passwd` command is one of the few that **must** be suid.

Setting suid and sgid

The suid and sgid bits are set and reset symbolically using the letter `s`; for example, `u+s` sets the suid access mode, and `g-s` removes the sgid mode. In the octal format, suid has the value 4 in the first (high order) digit, while sgid has the value 2.

Directories and sgid

When a directory has the sgid mode enabled, any files or directories created in it will inherit the group id of the directory. This is particularly useful for directory trees that are used by a group of people working on the same project. Listing 49 shows how user greg could set up a directory that all users of the xml-101 group could use, along with an example of how user gretchen could create a file in the directory.

Listing 49. sgid access mode and directories

```
greg@pinguino:~$ mkdir xml101
greg@pinguino:~$ chmod g+ws xml101
greg@pinguino:~$ ls -ld xml101
drwxrwsr-x  2 greg xml-101 6 Dec 25 22:01 xml101
greg@pinguino:~$ su - gretchen
Password:
gretchen@pinguino:~$ touch ~greg/xml101/gretchen.txt
gretchen@pinguino:~$ ls -l ~greg/xml101/gretchen.txt
-rw-r--r--  1 gretchen xml-101 0 Dec 25 22:02 /home/greg/xml101/gretchen.txt
```

Any member of group xml-101 can now create files in user greg's xml101 directory. As Listing 50 shows, other members of the group cannot update the file gretchen.txt, but they do have write permission to the directory and can therefore delete the file.

Listing 50. sgid access mode and file ownership

```
gretchen@pinguino:~$ su - tom
Password:
~$ cat something >> ~greg/xml101/gretchen.txt
-su: /home/greg/xml101/gretchen.txt: Permission denied
~$ rm ~greg/xml101/gretchen.txt
rm: remove write-protected regular empty file `/home/greg/xml101/gretchen.txt'? y
~$ ls -l ~greg/xml101
total 0
```

The sticky bit

You have just seen how anyone with write permission to a directory can delete files in it. This might be acceptable for a workgroup project, but is not desirable for globally shared file space such as the /tmp directory. Fortunately, there is a solution.

The remaining access mode bit is called the *sticky* bit. It is represented symbolically by `t` and numerically as a 1 in the high-order octal digit. It is displayed in a long directory listing in the place of the executable flag for other users (the last character), with the same meaning for upper and lower case as for suid and sgid. If set for a directory, it permits only the owning user or the superuser (root) to delete or unlink a file. Listing 51 shows how user greg could set the sticky bit on his xml101 directory and also shows that this bit is set for /tmp.

Listing 51. Sticky directories

```
greg@pinguino:~$ chmod +t xml101
greg@pinguino:~$ ls -l xml101
total 0
greg@pinguino:~$ ls -ld xml101
drwxrwsr-t  2 greg xml-101 6 Dec 26 09:41 xml101
greg@pinguino:~$ ls -ld xml101 /tmp
drwxrwxrwt  13 root root    520 Dec 26 10:03 /tmp
drwxrwsr-t   2 greg xml-101   6 Dec 26 09:41 xml101
```

On a historical note, UNIX® systems used to use the sticky bit on files to hoard executable files in swap space and avoid reloading. Modern Linux kernels ignore the sticky bit if it is set for files.

Access mode summary

Table 5 summarizes the symbolic and octal representation for the three access modes discussed here.

Table 5. Access modes

Access mode	Symbolic	Octal
suid	s with u	4000
sgid	s with g	2000
sticky	t	1000

Combining this with the earlier permission information, you can see that the full octal representation corresponding to greg's xml101 permissions and access modes of drwxrwsr-t is 1775.

Immutable files

The access modes and permissions provide extensive control over who can do what with files and directories. However, they do not prevent inadvertent deletion of files by the root user. There are some additional *attributes* available on various filesystems that provide additional capabilities. One of these is the *immutable* attribute. If this is set, even root cannot delete the file until the attribute is unset.

Use the `lsattr` command to see whether the immutable flag (or any other attribute) is set for a file or directory. To make a file immutable, use the `chattr` command with the `-i` flag.

Listing 52 shows that user root can create an immutable file but cannot delete it until the immutable flag is removed.

Listing 52. Immutable files

```
root@pinguino:~# touch keep.me
root@pinguino:~# chattr +i keep.me
root@pinguino:~# lsattr keep.me
----i------------ keep.me
root@pinguino:~# rm -f keep.me
rm: cannot remove `keep.me': Operation not permitted
root@pinguino:~# chattr -i keep.me
root@pinguino:~# rm -f keep.me
```

Changing the immutable flag requires root authority, or at least the CAP_LINUX_IMMUTABLE capability. Making files immutable is often done as part of a security or intrusion detection effort. See the capabilities man page (`man capabilities`) for more information.

The umask

When a new file is created, the creation process specifies the permissions that the new file should have. Often, the mode requested is 0666, which makes the file readable and writable by anyone. However, this permissive creation is affected by a *umask* value, which specifies what permissions a user does **not** want to grant automatically to newly created files or directories. The system uses the umask value to reduce the originally requested permissions. You can view your umask setting with the `umask` command, as shown in Listing 53.

Listing 53. Displaying octal umask

```
ian@pinguino:~$ umask
0022
```

Remember that the umask specifies which permissions should **not** be granted. On Linux systems, the umask normally defaults to 0022, which **removes** group and other write permission from new files. Use the `-S` option to display the umask symbolically, in a form that shows which are the permissions that **are** allowed.

You can use the `umask` command to set a umask as well as display one. So, if you would like to keep your files more private and disallow all group or other access to newly created files, you would use a umask value of 0077. Or set it symbolically using `umask u=rwx,g=,o=`, as illustrated in Listing 54.

Listing 54. Setting the umask

```
ian@pinguino:~$ umask
0022
ian@pinguino:~$ umask -S
u=rwx,g=rx,o=rx
ian@pinguino:~$ umask u=rwx,g=,o=
ian@pinguino:~$ umask
0077
ian@pinguino:~$ touch newfile
ian@pinguino:~$ ls -l newfile
-rw-------  1 ian ian 0 2005-12-26 12:49 newfile
```

The next section shows you how to change the owner and group of an existing filesystem object.

Section 7. Setting file owner and group

This section covers material for topic 1.104.6 for the Junior Level Administration (LPIC-1) exam 101. The topic has a weight of 1.

In this section, you learn about:

- Changing a file's group
- Default group for new files
- Changing the owner of a file

In the previous section you learned how every filesystem object has an owner and a group. In this section you learn how to change the owner or group of an existing file and how the default group for new files can be set.

File group

To change the group of a file, use the `chgrp` command with a group name and one or more filenames. You may also use the group number if you prefer. An ordinary

user must be a member of the group to which the file's group is being changed. The root user may change files to any group. Listing 55 shows an example.

Listing 55. Changing group ownership

```
ian@pinguino:~$ touch file1 file2
ian@pinguino:~$ ls -l file*
-rw-r--r--  1 ian ian 0 2005-12-26 14:09 file1
-rw-r--r--  1 ian ian 0 2005-12-26 14:09 file2
ian@pinguino:~$ chgrp xml-101 file1
ian@pinguino:~$ chgrp  1001 file2
ian@pinguino:~$ ls -l file*
-rw-r--r--  1 ian xml-101 0 2005-12-26 14:09 file1
-rw-r--r--  1 ian xml-101 0 2005-12-26 14:09 file2
```

As with many of the commands covered in this tutorial, `chgrp` has a `-R` option to allow changes to be applied recursively to all selected files and subdirectories.

Default group

In the previous section you learned how setting the sgid mode on a directory causes new files created in that directory to belong to the group of the directory rather than to the group of the user creating the file.

You may also use the `newgrp` command to temporarily change your primary group to another group of which you are a member. A new shell will be created, and when you exit the shell, your previous group will be reinstated, as shown in Listing 56.

Listing 56. Using newgrp to temporarily change default group

```
ian@pinguino:~$ newgrp xml-101
ian@pinguino:~$ groups
xml-101 adm dialout cdrom floppy audio dip video plugdev lpadmin scanner admin ian
ian@pinguino:~$ touch file3
ian@pinguino:~$ ls -l file3
-rw-r--r--  1 ian xml-101 0 2005-12-26 14:34 file3
ian@pinguino:~$ exit
ian@pinguino:~$ groups
ian adm dialout cdrom floppy audio dip video plugdev lpadmin scanner admin xml-101
```

File owner

The root user can change the ownership of a file using the `chown` command. In its simplest form, the syntax is like the `chgrp` command, except that a user name or numeric id is used instead of a group name or id. The file's group may be changed at the same time by adding a colon and a group name or id right after the user name or id. If only a colon is given, then the user's default group is used. Naturally, the `-R` option will apply the change recursively. Listing 57 shows an example.

Listing 57. Using newgrp to temporarily change default group

```
root@pinguino:~# ls -l ~ian/file4
-rw-r--r--  1 ian ian 0 2005-12-26 14:44 /home/ian/file4
root@pinguino:~# chown greg ~ian/file4
root@pinguino:~# ls -l ~ian/file4
-rw-r--r--  1 greg ian 0 2005-12-26 14:44 /home/ian/file4
root@pinguino:~# chown tom: ~ian/file4
root@pinguino:~# ls -l ~ian/file4
-rw-r--r--  1 tom xml-101 0 2005-12-26 14:44 /home/ian/file4
```

An older form of specifying both user and group used a dot instead of a colon. This is no longer recommended as it may cause confusion when names include a dot.

Section 8. Hard and symbolic links

This section covers material for topic 1.104.7 for the Junior Level Administration (LPIC-1) exam 101. The topic has a weight of 1.

In this section, you learn about:

- Hard links
- Symbolic links

Hard links

In the tutorial for topic 103, "LPI exam 101 prep (topic 103): GNU and UNIX commands," you learned that a file or directory is contained in a collection of *blocks* and that information about the file or directory is contained in an *inode*.

A *hard link* is a pointer to an inode. So, a file name is really a link to the inode that contains information about the file. As you learned, you can use the `-i` option of the `ls` command to display inode numbers for file and directory entries.

You can use the `ln` command to create additional hard links to an existing file (but not to a directory, even though the system sets up . and .. as hard links). If there are multiple hard links to an inode, then the inode is deleted only when the link count goes to zero.

Listing 58 shows how to create a file and then a hard link to it. It also shows that even though the original file name is removed, the second hard link prevents the inode from being erased when the first filename is removed.

Listing 58. Hard links

```
ian@pinguino:~$ echo testing > file1
ian@pinguino:~$ ls -l file*
-rw-r--r--  1 ian ian 8 2005-12-26 15:35 file1
```

```
ian@pinguino:~$ ln file1 file2
ian@pinguino:~$ ls -l file*
-rw-r--r--  2 ian ian 8 2005-12-26 15:35 file1
-rw-r--r--  2 ian ian 8 2005-12-26 15:35 file2
ian@pinguino:~$ rm file1
ian@pinguino:~$ ls -l file*
-rw-r--r--  1 ian ian 8 2005-12-26 15:35 file2
ian@pinguino:~$ cat file2
testing
```

Hard links may exist only within a particular filesystem. They cannot cross filesystems, since they refer to an inode by number, and inode numbers are only unique within a filesystem.

Finding hard links

If you need to find which files link to a particular inode, you can use the `find` command and the `-samefile` option with a filename or the `-inum` option with an inode number, as shown in Listing 59.

Listing 59. Finding hard links

```
ian@pinguino:~$ ln file2 file3
ian@pinguino:~$ ls -il file2
172 -rw-r--r--  2 ian ian 8 2005-12-26 15:35 file2
ian@pinguino:~$ find . -samefile file2
./file2
./file3
ian@pinguino:~$ find . -inum 172
./file2
./file3
```

Symbolic links

Another form of filesystem link that is used in Linux systems is a *symbolic link* (often called simply a *symlink*). In this case, the link refers to the name of another filesystem object rather than its inode. Symbolic links can refer to directories and can refer to files on other filesystems. They are frequently used to provide aliases for system commands. Using a long directory listing, you can see whether an object is a symbolic link when its first character is the lowercase letter `l`, as shown in Listing 60.

Listing 60. Symbolic link examples

```
ian@pinguino:~$ ls -l /sbin/mkfs.*
-rwxr-xr-x  1 root root  14160 2005-09-20 12:43 /sbin/mkfs.cramfs
-rwxr-xr-x  3 root root  31224 2005-08-23 09:25 /sbin/mkfs.ext2
-rwxr-xr-x  3 root root  31224 2005-08-23 09:25 /sbin/mkfs.ext3
-rwxr-xr-x  2 root root  55264 2005-06-24 07:48 /sbin/mkfs.jfs
-rwxr-xr-x  1 root root  13864 2005-09-20 12:43 /sbin/mkfs.minix
lrwxrwxrwx  1 root root      7 2005-12-14 07:40 /sbin/mkfs.msdos -> mkdosfs
-rwxr-xr-x  2 root root 241804 2005-05-11 09:40 /sbin/mkfs.reiser4
-rwxr-xr-x  2 root root 151020 2004-11-25 21:09 /sbin/mkfs.reiserfs
lrwxrwxrwx  1 root root      7 2005-12-14 07:40 /sbin/mkfs.vfat -> mkdosfs
-rwxr-xr-x  1 root root 303788 2005-04-14 01:27 /sbin/mkfs.xfs
```

In addition to the type of `l`, you can see on the right a `->` followed by the name that

the link refers to. For example, the `mkfs.vfat` command is a symbolic link to the `mkdosfs` command. You will find many other such links in /sbin and other system directories. Another tipoff is that the size is the number of characters in the link target's name.

You create a symlink using the `ln` command with the `-s` option as shown in Listing 61.

Listing 61. Creating symlinks

```
ian@pinguino:~$ touch file5
ian@pinguino:~$ ln -s file5 file6
ian@pinguino:~$ ln -s file5 file7
ian@pinguino:~$ ls -l file*
-rw-r--r--  2 ian ian 8 2005-12-26 15:35 file2
-rw-r--r--  2 ian ian 8 2005-12-26 15:35 file3
-rw-r--r--  1 ian ian 0 2005-12-26 17:40 file5
lrwxrwxrwx  1 ian ian 5 2005-12-26 17:40 file6 -> file5
lrwxrwxrwx  1 ian ian 5 2005-12-26 17:40 file7 -> file5
```

Note that the link counts in the directory listing are not updated. Deleting the link does not affect the target file. Symlinks do not prevent a file from being deleted; if the target file is moved or deleted, then the symlink will be broken. For this reason, many systems use colors in directory listings, often pale blue for a good link and red for a broken one.

Finding symbolic links

If you need to find which files link symbolically to a particular file, you can use the `find` command and the `-lname` option with a filename, as illustrated in Listing 62. Links may use a relative or absolute path, so you probably want a leading asterisk in the name to be matched.

Listing 62. Finding symbolic links

```
ian@pinguino:~$ mkdir linktest1
ian@pinguino:~$ ln -s ~/file3 linktest1/file8
.ian@pinguino:~$ find . -lname "*file3"
./linktest1/file8
ian@pinguino:~$ find . -lname "*file5"
./file7
./file6
```

Paths and symlinks

In most of the examples we have seen so far, the symlink has been in the same directory as the target, and the paths in the link have been implicitly relative paths. In Listing 62 we created a link in the linktest1 subdirectory, which used an absolute target (~/file3). When creating symlinks, you need to consider whether to use relative or absolute paths, since you may use either. Figure 3 illustrates the effect of moving a set of files and symlinks into a subdirectory.

Figure 3. Symlinks and paths

```
ian@pinguino:~$ mkdir linktest2
ian@pinguino:~$ ls file?
file2  file3  file5  file6  file7
ian@pinguino:~$ mv file? linktest2
ian@pinguino:~$ ls -l linktest1 linktest2
linktest1:
total 0
lrwxrwxrwx  1 ian ian 15 2005-12-26 18:07 file8 -> /home/ian/file3

linktest2:
total 8
-rw-r--r--  2 ian ian 8 2005-12-26 15:35 file2
-rw-r--r--  2 ian ian 8 2005-12-26 15:35 file3
-rw-r--r--  1 ian ian 0 2005-12-26 17:40 file5
lrwxrwxrwx  1 ian ian 5 2005-12-26 17:40 file6 -> file5
lrwxrwxrwx  1 ian ian 5 2005-12-26 17:40 file7 -> file5
```

The red color indicates that the linktest1/file8 link is now broken. This is not surprising as there is no longer a ~/file3. However, the two symlinks to file5 are still good as the file is still in the same relative location, even though it and the two links to it have moved. There are no hard and fast rules about whether to use relative or absolute paths in symbolic links; it depends somewhat on whether the link or the target is more likely to be moved. Just remember to consider the issue when making symbolic links.

Broken symlinks

One final point on our broken symbolic link. Attempts to read the file will fail as it does not exist. However, attempts to write it will work if you have the appropriate permission on the target file, as shown in Listing 63.

Listing 63. Reading from and writing to a broken symlink

```
ian@pinguino:~$ cat linktest1/file8
cat: linktest1/file8: No such file or directory
ian@pinguino:~$ echo "test file 8" >> linktest1/file8
ian@pinguino:~$ cat linktest1/file8
test file 8
ian@pinguino:~$ find . -name file3
./linktest2/file3
./file3
```

Since I can create files in my home directory, writing to the broken link created the missing target file.

Section 9. Finding and placing system files

This section covers material for topic 1.104.8 for the Junior Level Administration (LPIC-1) exam 101. The topic has a weight of 5.

In this section, you learn about:

- The Filesystem Hierarchy Standard and how files and directories are classified
- Finding files and commands

Filesystem Hierarchy Standard

The Filesystem Hierarchy Standard is a document that specifies the layout of directories on a Linux or other UNIX-like system. It was created to provide a common layout to simplify distribution-independent software development, by placing files in the same general place across Linux distributions. It is also used in the Linux Standard Base (see Resources).

The two independent FHS categories

At the core of the FHS are two independent characteristics of files:

Shareable vs. unshareable
: Shareable files can be located on one system and used on another, while unshareable files must reside on the system on which they are used.

Variable vs. static
: Static files include documentation, libraries, and binaries that do not change without system administrator intervention. Files that are not static are variable.

These distinctions allow files with different sets of characteristics to be stored on different filesystems. Table 6 is an example from the FHS document showing a layout that would be FHS-compliant.

Table 6. FHS example

	Shareable	Unshareable
Static	/usr /opt	/etc /boot
Variable	/var/mail /var/spool/news	/var/run /var/lock

The root filesystem

The FHS goal is to keep the root filesystem as small as possible. It must contain all the files necessary to boot, restore, recover, or repair the system, including the utilities that an experienced administrator would need for these tasks. Note that booting a system requires that enough be on the root filesystem to permit mounting of other filesystems.

Directories in the root

Table 7 shows the purpose of the directories that the FHS requires in the root (or /) filesystem. Either the directory or a symbolic link to it must be present, except for those marked optional, which are required only if the corresponding subsystem is present.

Table 7. FHS root filesystem

Directory	Purpose
bin	Essential command binaries
boot	Static files of the boot loader
dev	Device files
etc	Host-specific system configuration
lib	Essential shared libraries and kernel modules
media	Mount point for removable media
mnt	Mount point for mounting a filesystem temporarily
opt	Add-on application software packages
sbin	Essential system binaries
srv	Data for services provided by this system
tmp	Temporary files
usr	Secondary hierarchy
var	Variable data
home	User home directories (optional)
lib<qual>	Alternate format essential shared libraries (optional)
root	Home directory for the root user (optional)

/usr and /var

The /usr and /var hierarchies are complex enough to have complete sections of the FHS devoted to them. The /usr filesystem is the second major section of the filesystem, containing shareable, read-only data. It can be shared between systems, although present practice does not often do this. The /var filesystem contains variable data files, including spool directories and files, administrative and logging data, and transient and temporary files. Some portions of /var are not shareable between different systems, but others, such as /var/mail, /var/cache/man, /var/cache/fonts, and /var/spool/news may be shared.

To fully understand the standard, read the FSH document (see Resources).

Where's that file?

Linux systems often contain hundreds of thousands of files. The newly installed Ubuntu system that we have been using in this tutorial has nearly 50000 files in the /usr hierarchy alone. A Fedora system that I have been using for some time has

about 175000. The remainder of this section looks at tools to help you find files, particularly programs, in this vast sea of data.

Your PATH

If you have used several Linux systems, you may have noticed that if you log in as root, you are able to execute commands such as `fdisk`, which you apparently cannot execute if you are a user. What happens is that when you run a program at the command line, the bash (or other) shell searches through a list of directories to find the program you requested. The list of directories is specified in your PATH environment variable, and it is not uncommon for root's path to include /sbin, while non-root user paths do not. Listing 64 shows two different user path examples, as well as a root path example.

Listing 64. Some PATH examples

```
ian@pinguino:~$ echo $PATH
/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin:/usr/bin/X11:/usr/games
[ian@attic4 ~]$ echo $PATH
/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:/usr/X11R6/bin:/home/ian/bin
[ian@attic4 ~]$ su -
Password:
[root@attic4 ~]# echo $PATH
/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:/usr/local/bin:/sbin:/bin:
/usr/sbin:/usr/bin:/usr/X11R6/bin:/root/bin
```

As you can see, the PATH variable is just a list of directory names, separated by colons. Since the `fdisk` command is actually located in /sbin/fdisk, only the first and last of these paths would allow the user to run it by typing `fdisk` without providing a fully qualified name (/sbin/fdisk).

Usually, your path is set in an initialization file such as .bash_profile or .bashrc. You can change it for the current session by specifying a new path. Remember to export the PATH variable if you want the new value to be available to other processes that you start. An example is shown in Listing 65.

Listing 65. Changing your PATH

```
[ian@attic4 ~]$ fdisk
-bash: fdisk: command not found
[ian@attic4 ~]$ export PATH=/sbin:$PATH
[ian@attic4 ~]$ fdisk

Usage: fdisk [-l] [-b SSZ] [-u] device
E.g.: fdisk /dev/hda  (for the first IDE disk)
  or: fdisk /dev/sdc  (for the third SCSI disk)
  or: fdisk /dev/eda  (for the first PS/2 ESDI drive)
  or: fdisk /dev/rd/c0d0  or: fdisk /dev/ida/c0d0  (for RAID devices)
  ...
```

which, type, and whereis

In the previous example we discovered that the `fdisk` command was not available only by attempting to run it. There are several commands that can help you do this too.

which

You can use the `which` command to search your path and find out which command will be executed (if any) when you type a command. Listing 66 shows an example of finding the `fdisk` command.

Listing 66. Using which

```
[ian@attic4 ~]$ which fdisk
/usr/bin/which: no fdisk in (/usr/kerberos/bin:/usr/local/bin:/bin:/usr/bin:
/usr/X11R6/bin:/home/ian/bin)
[ian@attic4 ~]$ export PATH=/sbin:$PATH
[ian@attic4 ~]$ which fdisk
/sbin/fdisk
```

The `which` command shows you the first occurrence of a command in your path. If you want to know if there are multiple occurrences, then add the `-a` option as shown in Listing 67.

Listing 67. Using which to find multiple occurrences

```
[root@attic4 ~]# which awk
/bin/awk
[root@attic4 ~]# which -a awk
/bin/awk
/usr/bin/awk
```

Here we find the `awk` command in /bin (which contains commands that may be used by both the system administrator and by users, but which are required when no other filesystems are mounted) and also in /sbin (which contains the binaries essential for booting, restoring, recovering, and/or repairing the system).

type

There are some commands that the `which` command will not find, such as shell builtins. The `type` builtin will tell you how a given command string will be evaluated for execution. Listing 68 shows an example using the `type` command itself.

Listing 68. Using type

```
[root@attic4 ~]# which type
/usr/bin/which: no type in (/usr/kerberos/sbin:/usr/kerberos/bin:/usr/local/sbin:
/usr/local/bin:/sbin:/bin:/usr/sbin:/usr/bin:/usr/X11R6/bin:/root/bin)
[root@attic4 ~]# type type
type is a shell builtin
```

whereis

If you want to find more information than just the location of a program, then you can use the `whereis` command. For example, you can find the man pages or other information, as shown in Listing 69.

Listing 69. Using whereis

```
[root@attic4 ~]# whereis awk
awk: /bin/awk /usr/bin/awk /usr/libexec/awk /usr/share/awk
/usr/share/man/man1p/awk.1p.gz /usr/share/man/man1/awk.1.gz
```

Note that the copy of `awk` in /sbin was not found by `whereis`. The directories used by `whereis` are fixed, so the command may not always find what you are looking for. The `whereis` command can also search for source files, specify alternate search paths, and search for unusual entries. Consult the man pages to see how to override this behavior or change the fixed paths used by `whereis`.

find

In the tutorial for topic 103, "LPI exam 101 prep (topic 103): GNU and UNIX commands," you learned how to find files based on name (including wildcards), path, size, or timestamp. In the earlier section on Hard and symbolic links, you learned how to find the links to a particular file or inode.

The `find` command is the Swiss army knife of file searching tools on Linux systems. Two other capabilities that you may find useful are the ability to find files based on user or group name and the ability to find files based on permissions.

Listing 70 shows a directory listing for our sample workgroup directory ~greg/xml101, along with an example of how to find all the files owned by user ian, and all the ones that do not have the xml-101 group. Note how the exclamation point, `!`, negates the sense of a test when using `find`.

Listing 70. Finding files by user and group

```
ian@pinguino:~$ ls -l ~greg/xml101/*
-rw-r--r--  1 greg xml-101 0 2005-12-27 07:38 /home/greg/xml101/file1.c
-rw-r-----  1 greg xml-101 0 2005-12-27 07:39 /home/greg/xml101/file2.c
-rw-r--r--  1 tom  xml-101 0 2005-12-27 07:41 /home/greg/xml101/file3.c
-rw-r--r--  1 ian  ian     0 2005-12-27 07:40 /home/greg/xml101/file4.c
-rw-r--r--  1 tom  xml-101 0 2005-12-27 07:41 /home/greg/xml101/file5.c
-rw-r--r--  1 ian  xml-101 0 2005-12-27 07:40 /home/greg/xml101/file6.c
-rw-r--r--  1 tom  xml-101 0 2005-12-27 07:43 /home/greg/xml101/file7.c
-rwxr-xr-x  1 tom  xml-101 0 2005-12-27 07:42 /home/greg/xml101/myprogram
ian@pinguino:~$ find ~greg/xml101 -user ian
/home/greg/xml101/file4.c
/home/greg/xml101/file6.c
ian@pinguino:~$ find ~greg/xml101 ! -group xml-101
/home/greg/xml101/file4.c
```

To find files by permission, you can use the `-perm` test along with symbolic expressions similar to those used with the `chmod` or `umask` commands. You can search for exact permissions, but it is often more useful to prefix the permission

expression with a hyphen to indicate that you want files with those permissions set, but you don't care about other permissions. Using the files of Listing 70, Listing 71 illustrates how to find files that are executable by user and group, and two different ways of finding files that are not readable by others.

Listing 71. Finding files by permission

```
ian@pinguino:~$ find ~greg/xml101 -perm -ug=x
/home/greg/xml101
/home/greg/xml101/myprogram
ian@pinguino:~$ find ~greg/xml101 ! -perm  -o=r
/home/greg/xml101/file2.c
ian@pinguino:~$ find ~greg/xml101 ! -perm  -0004
/home/greg/xml101/file2.c
```

We have covered several major types of search that you can do with the `find` command. To further narrow your output, you can combine multiple expressions and you can add regular expressions to the mix. To learn more about this versatile command, use the man page, or better, use `info find` if you have the info system installed.

Listing 72 shows a final example of searching with `find`. This example does a cd to /usr/include to keep the listing length manageable, then finds all files containing `xt` in their path name without regard to case. The second example further restricts this output to files that are not directories and that are at least 2 kilobytes in size. Actual output on your system may differ depending on what packages you have installed.

Listing 72. A final example of find

```
ian@pinguino:/usr/include$ find . -iregex ".*xt.*"
./X11/xterm
./X11/xterm/ptyx.h
./irssi/src/fe-common/core/printtext.h
./irssi/src/fe-common/core/hilight-text.h
ian@pinguino:/usr/include$ find . -iregex ".*xt.*" ! -type d -size +2k
./X11/xterm/ptyx.h
./irssi/src/fe-common/core/printtext.h
```

Note that the regular expression must match the full path returned by find, and remember the difference between regular expressions and wildcards.

locate and updatedb

The `find` command searches all the directories you specify, every time you run it. To speed things up, you can use another command, `locate`, which uses a database of stored path information rather than searching the filesystem every time.

locate and slocate

The `locate` command searches for matching files in a database that is usually updated daily by a cron job. On modern Linux systems, this command is usually replaced by the `slocate` command, which stores permissions as well as paths and

thus prevents users from prying into directories that they could not otherwise see. On these systems you will usually find that `locate` is a symbolic link to `slocate`, so you can use either command.

The `locate` command matches against any part of a pathname, not just the file itself. Listing 73 shows that `locate` is a symlink to `slocate` and then shows how to find paths containing the string `bin/ls`.

Listing 73. Using locate

```
[ian@attic4 ~]$ ls -l $(which locate)
lrwxrwxrwx  1 root slocate 7 Aug 24 23:04 /usr/bin/locate -> slocate
[ian@attic4 ~]$ locate bin/ls
/bin/ls
/usr/bin/lsb_release
/usr/bin/lskatproc
/usr/bin/lspgpot
/usr/bin/lsattr
/usr/bin/lskat
/usr/bin/lshal
/usr/bin/lsdiff
/usr/sbin/lsof
/sbin/lsmod
/sbin/lsusb
/sbin/lspci
```

updatedb

The database used by `slocate` is stored in the /var filesystem, in a location such as /var/lib/slocate/slocate.db. If you see output such as Listing 74, then your system is not running this job.

Listing 74. No database for slocate

```
[ian@attic4 ~]$ locate bin/ls
warning: locate: could not open database: /var/lib/slocate/slocate.db: No such file or
directory
warning: You need to run the 'updatedb' command (as root) to create the database.
Please edit /etc/updatedb.conf to enable the daily cron job.
```

The database is created or updated using the `updatedb` command. This is usually run daily as a cron job. The file /etc/updatedb.conf is the configuration file for `updatedb`. To enable daily updates, the root user needs to edit /etc/updatedb.conf and set `DAILY_UPDATE=yes`. To create the database immediately, run the `updatedb` command as root.

Resources

Learn

- Review the entire LPI exam prep tutorial series on developerWorks to learn Linux fundamentals and prepare for system administrator certification.
- At the LPIC Program, find task lists, sample questions, and detailed objectives for the three levels of the Linux Professional Institute's Linux system administration certification.
- In "Basic tasks for new Linux developers" (developerWorks, March 2005), learn how to open a terminal window or shell prompt and much more.
- In the Proceedings of the Linux Symposium, Volume One (July 2005, Ottawa, Canada), learn more about ext3 in "State of the Art: Where we are with the Ext3 filesystem," a paper by M. Cao, et. al.
- XFS: A high-performance journaling filesystem is the home page of the XFS project at SGI.
- Reiser4 is the next version of the ReiserFS filesystem.
- qtparted is a graphical partitioning tool that uses the Qt toolkit.
- gparted is a graphical partitioning tool designed for the GNOME desktop; it uses the GTK+GUI library.
- The Linux Documentation Project has a variety of useful documents, especially its HOWTOs.
- The Quota mini-HOWTO can help answer questions on quotas.
- Visit the home of the Filesystem Hierarchy Standard (FHS).
- "Advanced filesystem implementor's guide, Part 3" tells you more about the tmpfs virtual memory filesystem.
- At the LSB home page, learn about The Linux Standard Base (LSB), a Free Standards Group (FSG) project to develop a standard binary operating environment.
- *LPI Linux Certification in a Nutshell* (O'Reilly, 2001) and *LPIC I Exam Cram 2: Linux Professional Institute Certification Exams 101 and 102 (Exam Cram 2)* (Que, 2004) are references for those who prefer book format.
- Find more tutorials for Linux developers in the developerWorks Linux zone.
- Stay current with developerWorks technical events and Webcasts.

Get products and technologies

- Order the SEK for Linux, a two-DVD set containing the latest IBM trial software for Linux from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- Download IBM trial software directly from developerWorks.

Discuss

- Participate in the discussion forum for this content.
- Read developerWorks blogs, and get involved in the developerWorks community.

About the author

Ian Shields
Ian Shields works on a multitude of Linux projects for the developerWorks Linux zone. He is a Senior Programmer at IBM at the Research Triangle Park, NC. He joined IBM in Canberra, Australia, as a Systems Engineer in 1973, and has since worked on communications systems and pervasive computing in Montreal, Canada, and RTP, NC. He has several patents and has published several papers. His undergraduate degree is in pure mathematics and philosophy from the Australian National University. He has an M.S. and Ph.D. in computer science from North Carolina State University. You can contact Ian at ishields@us.ibm.com.

Trademarks